PERSONALIZED PRINT

MARKUP LANGUAGE

PPML Templates: Methods and Workflows

Creating print-ready PPML document streams automatically and efficiently, directly from raw data

Functional Specification

PPML Templating Specification

Version 1.0 – December 12, 2002

The PPML Working Group

 http://www.podi.org

the Digital Printing initiative

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, and related specifications, are 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: http://www.podi.org

Table of Contents

Chapter 1: Introduction 1
1.1 Purpose 1
1.2 Prerequisite reading 1
1.3 PPML as part of a larger workflow 1
1.4 Scope of this specification 3
1.5 Notation used in this document 3
1.6 Definitions 4
1.7 Requirements 5
Chapter 2: Applications of PPML Templating 7
2.1 Introduction 7
2.2 Benefits of templating 7
2.3 How templating differs from conventional workflows 7
2.4 Examples 10
Chapter 3: XML, Scripting, and XSLT 13
3.1 Introduction 13
3.2 Scripting technologies for PPML Templates 13
3.3 XML 13
3.4 XSLT 13
3.5 Format of an XSL template file 14
3.6 Literal Result Element as Stylesheet 15
3.7 General sequence of events in XSL 15
Chapter 4: Structure of a PPML Templating Project 17
4.1 Overview 17
4.2 Element content: Internal vs. External Data 18
4.3 The <PPMLT> Element 19
4.4 The <TEMPLATE> Element 20
4.5 The <TEMPLATE_REF> Element 22
4.6 The <DATA> Element 23
4.7 The <DATA_REF> Element 24
4.8 The <EXTERNAL_DATA> Element 25
4.9 The <INTERNAL_DATA> Element 26
4.10 The <DATA_STRUCTURE> Element 27
4.11 The <DATA_MAPPER> element 28
4.12 The <DATA_MAPPER_REF> Element 30
4.13 The <INPUT_DATA_STRUCTURE> Element 31

4.14 The <OUTPUT_DATA_STRUCTURE> Element....32

Chapter 5: Data....33

5.1 Introduction....33

5.2 The <RECORDS> element....35

5.3 The <R> element....36

5.4 The <F> element....37

5.5 Very long data streams....38

Appendix A: Sample Application....41

A.1 Introduction....41

A.2 Example 1: PPML Templating code, including Reusable Object definitions, complete PPML Template and Data Mapper, and data records....41

A.3 The same dataset, if the Reusable Object occurrences were defined and downloaded earlier....49

A.4 Leanest form: Template, Reusable Content, and Data Mapper have all been downloaded in advance....53

A.5 Example results....55

Chapter 1: Introduction

1.1 Purpose

The purpose of this specification is to enable PPML workflows that have higher value, by automating the generation of certain document streams directly from raw data, and to encourage growth of high-value workflows by defining an industry standard way of doing it.

The motivation for PPML templating is to enable very long runs of the most valuable type of digital printing – personalized content – without the substantial overhead traditionally required to generate and transmit large amounts of repetitive data. As shown by the examples in the Appendix, when properly applied PPML templating can reduce the amount of data required for such a print run by two or three orders of magnitude.

Not all PPML applications are suited to templating. PPML is capable of a very wide range of digital print applications; templating is appropriate for those where parts of the PPML stream can be replaced with variable content as described in this document.

1.2 Prerequisite reading

This document presumes that the reader is familiar with the basic concepts of PPML, the Personalized Print Markup Language. Readers who don't know PPML can use this document to learn the basic idea of templating. But to create actual templated projects, it's necessary to fully understand PPML. The PPML specification is freely available at http://www.PODi.org.

1.3 PPML as part of a larger workflow

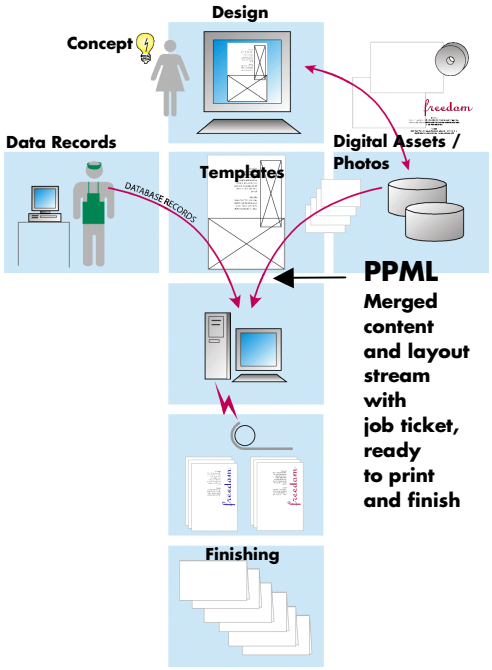
1.3.1 Introduction

PPML is the Personalized Print Markup Language, an XML-based metalanguage for digital print applications. Developed by the members of PODi, the Digital Print initiative, PPML was introduced to the market in February 2000 and has received statements of support from almost all vendors of digital print equipment and related application software. For more information see http://www.podi.org.

A major strength of PPML is its XML syntax. This means the entire range of XML data processing tools can be used to generate, analyze, and process PPML data. PPML templating, as described in this document, takes advantage of this.

1.3.2 The PPML 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.



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

- 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.

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. Today PPML 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.** As defined in this document, 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.
- **A design for packaging content and job ticket for reliable transport.** An appendix to the PPML specification defines VDP 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 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.

Further extension of the PPML architecture is being planned. For more information, contact PODi at info@podi.org.

1.4 Scope of this specification

This specification describes an extension to PPML for use with scripting technologies such as XSLT. It does not fully describe XSLT, nor the PPML language itself. Rather, it only describes a method of accomplishing the transform into PPML.

1.5 Notation used in this document

The following typographic notation is used in this document.

- PPML code excerpts, element names, and attributes: `Letter Gothic`
- XML code samples are shown as displayed and formatted in the XMLSpy Integrated Development Environment, for instance:

```
<!-- ================================== -->
<!-- Sheet Order and Face Up/Down choices -->
<!-- ================================== -->
<DigitalPrintingParamsUpdate UpdateID="SameOrderFaceUp" PageDelivery="SameOrderFaceUp"/>
<DigitalPrintingParamsUpdate UpdateID="SameOrderFaceDown" PageDelivery="SameOrderFaceDown"/>
<DigitalPrintingParamsUpdate UpdateID="ReverseOrderFaceDown" PageDelivery="ReverseOrderFaceDown"/>
<DigitalPrintingParamsUpdate UpdateID="ReverseOrderFaceUp" PageDelivery="ReverseOrderFaceUp"/>
```

  (In the body of this specification, no special formatting is applied to JDF element names.)

- 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 Courier, 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`."

1.6 Definitions

1.6.1 General PPML-related terms

Chapter 3 of the PPML Specification, "Terminology and Basic Concepts," defines basic terms regarding PPML document structure and workflow, including:

- **PPML Producer** (or simply "Producer") is anything that generates PPML files. This may be a standalone application, a system-level driver, or anything else.
- **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. See the PPML specification for details on Consumer functionality. In particular, note that not all Consumers are required to support all features.

  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.
- **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.
- **Dataset:** a `PPML` element, typically containing one or more Jobs and/or Reusable Object definitions and related elements required to process them.
- **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.

  NOTE: the PPML language includes a `<JOB>` element with specific meaning in the hierarchical structure of PPML. (In PPML 2.1 `DOCUMENT_SET` is preferred; its meaning is identical.) In this document, "Job" (capitalized) refers to a PPML `<JOB>` element; "job" (lowercase) is informal, with no special meaning relative to PPML. For instance, it's correct to say "The supervisor asked Pat to run job #482, which contained three PPML Jobs."

The following terms are also used in this document:

- **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. See Chapter 6 of the PPML Specification.
- **Print Originator:** the person (or group) for whom a project is being produced – the person who conceived it and/or decided what they want the finished result (Product) to look like. The print originator knows what the desired end product is, and may have no knowledge of process, i.e. how the job will be produced.
- **Production Shop:** traditionally this term refers to the people who produce a print job, including their equip ment, software, procedures, and the physical facility itself. This may be a department of a company, a separate business, or any other entity. In this document, the term

refers to any entity that performs the work of such a Production Shop. See also "Workflow" under JDF, below.

- **RIP:** Raster Image Processor – an image processing system that reads data expressed in a PDL such as PostScript® and converts it to a raster image.
- **Format Processor:** the component of a PPML Consumer that processes objects submitted in a particular input data format. Examples: a PostScript RIP, or a module that can directly process an image format such as JPEG or TIFF.
- **PPML Job Ticket:** the data required to produce a set of printed documents, beyond the document content and layout specified in PPML. The ticket may be external to the PPML dataset or may be embedded in it, within a PPML `TICKET` element.

1.6.2 Terms related to Templating

- **PPML Template Producer** (or simply "Template Producer") is anything that generates a PPML Template. This may be a human editing a file, a standalone application, or anything else.
- **PPML Template Consumer** (or simply "Template Consumer") is anything that reads and executes `PPMLT` elements as defined in this document. PPML Template Consumers are PPML Producers as defined above: they generate PPML code.

  It is expected, but not required, that many PPML Consumers will also include built-in Template Consumer capability. But the Template Consumer may also be a separate production step, physically distinct from the PPML Consumer. In that case, the Template Consumer would read the PPMLT elements and produce an ordinary stream of PPML documents, which the PPML Consumer would process the same as if they didn't come from templating.
- **Template:** In the context of PPML Templating, the template is a prototype PPML document which has been modified to enable varying the content using a scripting language such as XSLT.
- **Data records:** the input data that will be merged with the template to generate the stream of Instance Documents. Example: the recipient's name, address, and last product purchased.
- **Style sheet:** In the context of PPML Templating this refers to XSLT style sheets. An XSLT style sheet contains the transformation instructions that define how Data Records are merged with the Template. For further information, see the XSLT web site.

1.7 Requirements

The PPML Templating workflow was developed to meet the following requirements:

1.7.1 Must not interfere with existing PPML Consumers in non-template applications

1.7.2 Support multiple formats for the data list

- XML, both simple and complex

- Comma-separated
- Line data

1.7.3 Compatible with PPML Requirements

PPML Templating shall not restrict or interfere with any existing requirements for the PPML datastream itself. In particular, it shall follow the PPML requirements for streaming; page independence, locatability, segmentability; manufacturing information; and variable document length.

1.7.4 Flexible workflow: Allow template and data to be transmitted in a single package or separately, which enables reusability of the data and the template

- Allow sending just a template to the Template Consumer for later use
- Allow sending just the data, to be used with a previously sent template. Note that the same template can be used repeatedly with different sets of data. This, in fact, is the most productive way to use variable data, since it amortizes the project's initial setup cost over a much longer project lifespan.

Chapter 2: Applications of PPML Templating

2.1 Introduction

There is a range of possibilities for how the processing of a print job can be divided among system components to achieve the benefits of a templating workflow. This chapter presents a framework for understanding the choices, and the benefits and limitations of what templating can do, so that workflow designers can make well-informed choices and make optimal use of this technology for their chosen applications.

In general the amount that can be achieved through templating depends on the amount of information available to the PPML Template Producer.

For instance, typically a template processor will be very good at performing routine, iterative operations like inserting data into a pre-defined layout. But features like text composition and detection of reusable content may not be part of a Template Consumer's abilities, in which case they would need to be handled elsewhere in the workflow, before data is transmitted to the Template Consumer. The PPML Template Producer may or may not have access to information generated during those processes, such as the height of copy blocks after they're composed.

2.2 Benefits of templating

PPML Templating involves downloading as much as possible of a personalized print project before the production run begins. PPML itself offers significant efficiencies in file size, and templating carries it even further: it takes advantage of the fact that for many print projects, much of the print stream is repetitive and can be stored in the digital printing press (the PPML Consumer).

In a fully optimized PPML Template workflow, virtually nothing remains to be downloaded at print time except the data itself. Very little data is generated, very little data is transmitted, and very little data is processed at the receiving end. As shown by the examples in the appendix, the result can be substantial savings.

In addition, by directly transforming variable data into a structured print stream (PPML), PPML Templating enables new workflows independent of constraints of traditional graphic arts system.

2.3 How templating differs from conventional workflows

2.3.1 Anatomy of a variable print project

To understand the impact of templating, it's useful to understand the environment in which it's designed to be used: the generation of streams of personalized documents.

Templating involves a simple but elegant shift in the location of one portion of the personalized print workflow: the point where the variable data is merged with the page layout.

As suggested above, personalized documents are typically characterized by having a common and repetitive structure, with content that varies from individual to individual. There are many different ways to generate such a stream. Examples range from simple office mail-merge (e.g. form letters) to highly sophisticated, automated, multi-variable page design systems which can completely vary the content and even the page layout. Regardless of the details, all these workflows must, at one time or another, accomplish two primary tasks:

- **Job setup**, in which the form letter or other document is designed
- **Production runs**, in which batches of documents are generated and printed.

More specifically, all personalized print workflows must handle the following tasks.

- **Job Setup**
    - **Design of the basic document** that will be personalized: decisions about content and layout.
    - **Decisions about how layout and content will vary**, depending on the variable data. These decisions are often referred to in VDP applications as rules. The rules are initially created by humans, and they must be encoded in some electronic form and stored somewhere, to enable automated production.
    - **Creation of the reusable content objects** and downloading to the print system.
- **Production runs**
    - **Selecting the variable data** for each print run. *This is unchanged in Templating.*
    - **Executing the VDP rules** for each recipient, to generating the personalized documents. *In Templating, this is done at a different time and place.*
    - **Generating the output code** to make the target print system produce the documents. *In PPML templating, this is merged with the previous step.*

The follow sections describe typical workflows, with and without templating. Many variations on these workflows exist in current practice in the industry; the purpose of this discussion is to illustrate the change that PPML Templating creates.

2.3.2 Conventional Variable Data workflow

- **Job setup:**
    - The page producer application ("Producer") generates:
        - A document template including
            - The basic document, including designation of what areas may change. This is typically stored in the Producer's native format.

        - The rules for creating instance documents, e.g. where to insert variable text and how to select reusable content elements. Like the layout, the rules are typically stored in the Producer's native format.
        - The definitions of PPML Reusable Objects
    - The Reusable Object definitions are typically transmitted to the Consumer for processing, proof-printing, approval, and storage.
- **Production runs:**
    - The database generates the variable data records for a print run. For instance, this may include the name, address, and product interest of each recipient.
    - The Producer opens the stored template and merges it with the variable data for each variable data record, using the template's VDP rules. The result is a stream of personalized documents.
    - The Producer uses its PPML driver to convert the personalized documents to a stream of PPML Instance Documents.
    - The PPML Instance Documents are transmitted from the Producer to the Consumer. (This may be done in many ways: by direct cable connection from Producer to Consumer, or by copying the PPML dataset to a CD, or compressing it into an archive such as a Zip file, etc.)
    - The Consumer executes the PPML, resulting in a stream of printed documents.

The next section illustrates the similarities – and differences – in a PPML Templating workflow.

2.3.3 PPML Template workflow

The initial document design is identical and the end result (the printed output) is identical. But several of the steps are moved to a different point in the workflow.

- **Job setup** is very similar to the conventional workflow, but the VDP rules are encoded in a scripting language (e.g. XSLT syntax) in a prototype PPML document. (See next chapter for more information on XSLT.) The template is downloaded to the PPML Template Consumer.

  Note: "PPML Template Consumer" refers to the part of the workflow that executes the template. This functionality may be resident inside the PPML Consumer, or it may take place in a pre-processing step. This has no bearing on the content of this specification.
- **Production:**
    - The database generates the same variable data records for a print run. The database records are packaged in the `DATA` element described in this specification.
    - The data is transmitted to the Template Consumer.

- *New:* The Template Consumer opens the transmitted PPML Template dataset, accesses the template (which was sent earlier), and executes the template's instructions once for each variable data record. The result is a stream of PPML Instance Documents. Note:
  - Functionally, this merging of template and data is the same as what the Producer did in the conventional workflow: the VDP rules are applied to each recipient's data. But in this workflow, the rules are executed inside the Template Consumer.
  - With this workflow it is not necessary for the Producer to generate an intermediate stream of personalized documents in its native format, and then convert each one to PPML for transmission to the PPML Consumer. Instead, the PPML is generated directly from the raw data. Also, if the PPML Template is processed inside the PPML Consumer, it avoids redundant generation and transmission of the basic document structure.
- At this point the stream of PPML code looks exactly the same as it did in the conventional workflow described above.
- Thus, the print stream that the PPML Consumer executes is identical in both workflows, and the resulting output is identical. But the result was produced with a bare minimum of data handling.

2.4 Examples

Templating can offer significant savings in processing on the originator side, and in data volume in the overall workflow, but there are some situations in which it may not be appropriate, or in which more sophisticated work may be necessary to create the transformations, involving more advanced applications of scripting technologies. This section presents some issues for consideration by designers of templating workflows.

The principle that "the amount templating can accomplish is a function of how much data is sent" is illustrated by the following discussion of basic and advanced applications.

2.4.1 Basic applications

In the simple case, PPML Templating is best suited to applications where the layout, page count, line breaks, and reusable objects are known in advance, so they don't need to be computed as part of executing the template. Examples include (but are by no means limited to):

- Simple direct mail pieces, like a "mail merge" application, including variable text or images.
- Point-of-purchase materials, using PPML reusable content with variable pricing etc.
- Photo album pages, with a static layout and a reusable page border but with references to different JPEG files on every page.

2.4.2 Advanced Applications

Advanced workflow developers, with strong knowledge of both PPML and a scripting technology, can create highly complex and sophisticated documents with this technology. For example:

- Personalized catalogs with variable sized items on different pages
- Customized financial statements that include data-driven graphics and individually selected offers or advisory content
- Collateral on demand, such as brochures assembled on the fly in response to a user's selections on a Web site.

2.4.3 Considerations in the design of templating workflows

As shown by the above examples, a broad range of workflows can be designed using PPML Templating, by choosing appropriate tools and components based on the type of output desired and the data available for input to the process. Factors to consider include:

- How will the workflow manipulate and transform the data? (For instance, can a particular scripting language accomplish the necessary conversions?)
- How will the workflow convert the raw uncomposed data into objects that can be placed on pages using PPML? (For instance, if text needs to be reflowed into composed paragraphs, how will that be achieved?)

For assistance in configuring workflows for a particular need, consult your vendor of PPML Templating systems.

Chapter 3:
XML, Scripting, and XSLT

3.1 Introduction

This chapter presents an introduction to scripting, particularly XML and XSLT, as they apply to PPML templating. For detailed information, see the resources listed here.

3.2 Scripting technologies for PPML Templates

This specification defines an architecture that can be used with any sort of scripting technology. As described below, XSLT is a natural choice, because it is based on XML, as is PPML itself.

However, PPML Templating applications are not limited to XSLT. For instance, another language that is expected to be well-suited to templating is PERL.

3.3 XML

PPML is an application of XML, the Extensible Mark-up Language. Readers who are not familiar with XML are directed to these resources:

- 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.

3.4 XSLT

3.4.1 Overview of what XSLT does

XSLT http://www.w3.org/TR/xslt is one of the two parts of XSL, the Extensible Style Language http://www.w3.org/Style/XSL. It transforms one XML file into another one. For instance, it can be used to transform data conforming to one DTD into a form conforming to another.

The expression language of XSLT is XPath.

3.4.2 High-level description

An XSLT processor does these things:

- It reads a set of instructions encoded in a "style sheet" file [.xsl] and builds a tree representation in memory. The term "style sheet" is historical. Although XSLT can indeed be used to insert formatting instructions, it is actually a general purpose XML transformation language.
- The style sheet includes two general things:
  - XSLT instructions, which tell the XSLT processor what data to locate in the input file(s) and how to manipulate and interpolate that data into the output
  - Literal text and non-XSLT XML
- The processor reads the default XML source file and builds a tree representation in memory. As processing continues, other source XML files may be opened but there is always one (and only one) available at the outset.
- Instructions in the style sheet tree direct the processor to generate a result tree in memory consisting of nodes created by copying the literal text and non-XSLT XML plus the result of executing other XSLT instructions. These other XSLT instructions create what are known as "result tree fragments", or RTFs, which are subsequently copied to the end result tree.
- Optionally, the processor serializes the result tree into an output file. The format of the output file may or may not be XML depending upon the output method specified in the XSLT script. Typical output methods are "xml", "html" and "text". Unless otherwise specified, the output method will be "xml".

There are many XSL processors. A commonly used one is Xalan (http://xml.apache.org/xalan-j/index.html). Saxon (http://saxon.sourceforge.net/) is another very popular XSL processor. A good list of XSL software tools is at http://xml.coverpages.org/xslSoftware.html

3.5 Format of an XSL template file

An XSL template file, or script, is a well formed XML file consisting of:

- XSLT instructions
- Non-XSLT XML
- Plain text

The non-XSLT XML and plain text constitute what are known as "Literal Result Elements" because they are copied to the result tree unmodifield.

One of the common XSLT instructions is `xsl:template`. The `xsl:template` instruction can be invoked by name or, more commonly, by matching some construct in the input XML file or files. The match is described using the XPath (http://www.w3.org/TR/xpath) XML vocabulary.

Each `xsl:template` itself contains XSLT instructions, non-XSLT XML and plain text. This content is processed when the template matches a construct in the input XML.

Another common XSLT instruction is `xsl:apply-templates`. The `xsl:apply-templates` instruction selects a set of nodes from the input XML and looks for the `xsl:template` instruction that best matches each, if any. That `xsl:template` instruction is then executed with the matching source node as an implicit parameter.

Every source XML file has an implicit, anonymous node at its head. This node, known as the "root node," is the parent of the top-level node in the source XML document. This node would be matched by an XPath match description of "/".

3.6 Literal Result Element as Stylesheet

A special form of XSLT scripts is known as a "Literal Result Element as Stylesheet". In this special form, no `xsl:template` instructions are present. Rather, an arbitrary XML file can have the XSLT namespace defined on it and be passed to an XSLT processor as the template, or script, file.

The entire XML file is treated as if it were a single Literal Result Element located within an `xsl:template` that matched the root node, i.e., match="/". As such, any non-XSLT XML and its plain text content are copied to the result tree verbatim. Any XSLT instructions found are evaluated as encountered and the outcome is also placed in the result tree.

By associating the XSLT namespace with a PPML document, it is possible to add XSLT instructions and pass that document as a stylesheet to an XSLT processor.

3.7 General sequence of events in XSL

This describes the general sequence, which will be illustrated by the example below.

- The XSL processor program is started with three parameters:
  - The initial input XML data file (e.g. the customer records)
  - The template file to read (the XSL instructions for how to generate an output file)
  - The name of the output file to create

  Sample command line (using the Xalan processor `xsl.exe`):

```
xsl  mydata.xml  ppml.xsl  ppmlout.xml
```

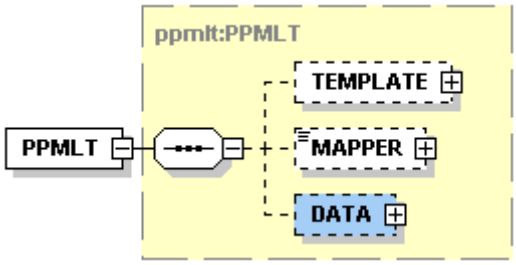
- The processor begins by reading the template file and building a tree representation of that file in memory. Note that the template file must be a well-formed XML file.
- The processor then reads the source XML file (the variable data records), which also must be well-formed, and builds a tree representation of that file in memory.
- The processor searches for the xsl:template that matches the XPath expression "/", i.e., the root node, and applies that template to the node. If no such template exists, the default template is one that applies templates to the children of the current node, recursively. As described above, an XSLT file employing a Literal Result Element as Stylesheet is considered to be the content of a template matching the root node.

- Any non-XSLT data, text or XML, is copied to the result tree.
- Upon encountering an `<xsl:...` instruction (these are highlighted in blue in the examples in this document), the instruction is evaluated and the result is placed on the result tree. These instructions may locate data in the source XML file, manipulate that data or both.

Chapter 4: Structure of a PPML Templating Project

4.1 Overview

A PPML Templating project requires two components for `TEMPLATE` and `DATA`. An optional third element, `DATA_MAPPER`, may connect them:



- The `TEMPLATE` element contains a special kind of PPML document that is ready to be merged with variable data as described in this specification.
- The `DATA` element contains the variable data records for one particular print run. The most common formats for the data are expected to be XML and comma-separated values.
- An optional `DATA_MAPPER` element can define data format conversions.

It is expected that these job components will often be delivered from Producer to Consumer in a single package (see the PPML specification's "packaging" appendix) but that process is not required. The template and data may be created and delivered at any time, independent of each other.

4.2 Element content: Internal vs. External Data

As described in the PPML Specification, PPML document content and resources may be contained directly in the XML element that requires them, or they may be external, identified by a URI reference. For PPML Templating, this method is used for the `TEMPLATE` element and the `DATA` element. Below are simple examples of a `TEMPLATE` element constructed both ways.

4.2.1 External Data example (reference to an external file)

```
<TEMPLATE ...>
  <EXTERNAL_DATA  Src="Template472.xml" />
</TEMPLATE>
```

External data may or may not be included in the same PPML Template package. See Appendix D of the PPML Specification for discussion of the advantages of delivering a complete package, and for recommended restrictions on the value of the Src attribute to ensure cross-platform portability.

In applications where the PPML Template Producer wants to ensure that a specific version of the referenced data is used, the Producer can provide a checksum.

4.2.2 Internal Data example

The Internal Data method inserts the content of the template into the `TEMPLATE` element, verbatim.

```
<TEMPLATE ...>
  <INTERNAL_DATA>
    (The template data goes here – the content of the file "Template472.xml"
     that was referenced by EXTERNAL_DATA in the example above)
  </INTERNAL_DATA>
</TEMPLATE>
```

4.2.3 Example of referencing content downloaded earlier

Content may also be downloaded and saved with a name for reference later. Example:

```
<TEMPLATE Name="472" Environment="TestFiles">
  <EXTERNAL_DATA  Src="Template472.xml" />
</TEMPLATE>
```

The content can later be referenced as follows:

```
<TEMPLATE_REF Name="472" Environment="TestFiles"/>
```


4.3 The <PPMLT> Element

4.3.1 Description

The `PPMLT` element is the top level, encompassing all components of the PPML templating job or this portion of it. (For instance, the template may be transmitted in one `PPMLT` element, and the data may be transmitted in a separate `PPMLT` element.)

4.3.2 Model

```
PPMLT        (
                (DATA
                    |  TEMPLATE, ((DATA_MAPPER | DATA_MAPPER_REF)?,
                          (DATA | DATA_REF))?
                    |  TEMPLATE_REF, ((DATA_MAPPER | DATA_MAPPER_REF)?,
                          (DATA | DATA_REF))
                    |  DATA_MAPPER
                )
             )
```

4.3.3 Attributes

None.

4.3.4 Result of processing a PPMLT element

A `PPMLT` element can contain, directly or by reference, a template or data or both. The expected behavior of the Template Consumer for each case is:

4.3.4.1 Template only

Extract the template data from the PPMLT (the INTERNAL_DATA content or the referenced EXTERNAL_DATA) and save it locally in the Template Consumer.

4.3.4.2 Data only

Extract the data from the PPMLT `DATA` element (specifically, from the `INTERNAL_DATA` content or the referenced `EXTERNAL_DATA` inside the `DATA` element) and save it locally in the Template Consumer.

4.3.4.3 Both template and data are identified

Execute the specified template, operating on the specified input data.

4.4 The <TEMPLATE> Element

4.4.1 Description

The TEMPLATE element identifies the prototype PPML document which will be used to generate the PPML Instance Documents.

The TEMPLATE element can contain either an INTERNAL_DATA or an EXTERNAL_DATA element. This means new template instructions (layout and VDP rules) can be downloaded within the PPML Template instance (INTERNAL_DATA) or the template can be downloaded in advance and referenced with EXTERNAL_DATA.

Allowing the template to be either internal or external to the dataset provides flexibility that can be useful in a variety of situations. See examles below.

An optional DATA_STRUCTURE element can be included, to describe the structure of the variable data expected by the template. This description can be used to allow validation by the receiving system or by any intermediate processing tools. (Note that the data contained in the DATA element may not be XML; if not, it must be converted to XML before merging with the template. See discussion below.)

4.4.2 Model

```
TEMPLATE     (DATA_STRUCTURE?, (INTERNAL_DATA | EXTERNAL_DATA))
```

4.4.3 Attributes

Attribute	Required /Optional	Type	Description
Format	Required	String	The format of this template. Must be a valid MIME type.
Name	Optional	String	Name to be used when referring to this template. The name must be unique within the template's environment. If this attribute is used, the Template Consumer must save this template, making it available for reference by subsequent PPMLT elements via a TEMPLATE_REF element.
Environment	Optional	String	Specifies the environment in which the template's name exists. (There is no default environment.) Required if the Name attribute is used.

4.4.4 Context

The TEMPLATE element occurs only within a PPMLT element.

4.4.5 Application note: saving templates for later use

When a PPML Template Consumer processes a TEMPLATE element with a Name attribute (and therefore also with an Environment attribute), the Template Consumer is required to save the template for later use, along with the value of the Format attribute.

Management of templates installed in a Template Consumer is not specified in this document. The developer of a Template Consumer is responsible for providing a way to manage them, e.g. deleting templates that are no longer needed.

4.4.6 Examples

Example 1: `TEMPLATE` element contains an External Data reference. This method would typically be used in case of repetitive projects, in which the template stays resident at the receiving system; in such cases, the PPML Template dataset transmits the variable data (in the `DATA` element) and references the previously downloaded template file.

```
<PPMLT>
   <TEMPLATE Format="text/xslt+xml">
     <EXTERNAL_DATA  Src="Project412.xsl"/>
   </TEMPLATE>
   <DATA>
     <INTERNAL_DATA>
        Variable data records go here
     </INTERNAL_DATA>
   </DATA>
</PPMLT>
```

Example 2: The template is contained in an `INTERNAL_DATA` element; the `DATA` element references data that was downloaded earlier. This approach could be used to generate a different document stream from a set of variable data records that were already sent earlier.

```
<PPMLT>
   <TEMPLATE>
     <INTERNAL_DATA>
        Template data goes here
     </INTERNAL_DATA>
   </TEMPLATE>
   <DATA>
     <EXTERNAL_DATA  Src="Project412_2003-10-09.xml"/>
   </DATA>
</PPMLT>
```

Example 3: `TEMPLATE` and `DATA` elements both contain External Data references. The following is a complete PPML Templating file, sufficient to cause the printing of a batch of personalized documents by associating a previously downloaded template with a previously downloaded variable data file.

```
<PPMLT>
   <TEMPLATE>
     <EXTERNAL_DATA  Src="Project412.xsl"/>
   </TEMPLATE>
   <DATA>
     <EXTERNAL_DATA  Src="Project412_2003-10-09.xml"/>
   </DATA>
</PPMLT>
```

4.5 The <TEMPLATE_REF> Element

4.5.1 Description

The `TEMPLATE_REF` element identifies, by reference, a template that has already been installed in the Template Consumer using a `TEMPLATE` element with the `Name` and `Environment` attributes.

4.5.2 Model

```
TEMPLATE_REF Empty
```

4.5.3 Attributes

Attribute	Required /Optional	Type	Description
Ref	Required	String	Name of the previously installed template. The name must be unique within the template's environment.
Environment	Required	String	Specifies the environment of the template's name. (There is no default environment.)
Checksum	Optional	String	Hexadecimal-encoded string, provided as a hint to the Template Consumer as an aid in identifying the template that was installed earlier. Template 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".

4.5.4 Context

The `TEMPLATE_REF` element occurs only within a `PPMLT` element.

4.6 The <DATA> Element

4.6.1 Description

The DATA element contains the database records to be merged with the template, to generate personalized Instance Documents. For a deeper discussion of handling different types of data, see Chapter 5: Data.

4.6.2 Model

```
DATA        (DATA_STRUCTURE?, (INTERNAL_DATA | EXTERNAL_DATA))
```

4.6.3 Attributes

Attribute	Required /Optional	Type	Description
Format	Required	String	The format of this data. Any valid MIME type.
Name	Optional	String	Name to be used when referring to this data. The name must be unique within the template's environment.
Environment	Optional	String	Specifies the environment in which the template's name should be available. (There is no default environment.) Required if the Name attribute is used.

4.6.4 Context

The DATA element occurs only within a PPMLT element.

4.6.5 Application note: saving data files for later use

When a PPML Template Consumer processes a DATA element with a Name attribute (and therefore also with an Environment attribute), the Template Consumer is required to save the data for later use, along with the value of the Format attribute.

Management of data files installed in a Template Consumer is not specified in this document. The developer of a Template Consumer is responsible for providing a way to manage them, e.g. deleting files that are no longer needed.

4.6.6 Character set conversion

The PPML Template Consumer may need to convert the data from the character set specified in the enclosed data element to the character set expected by the Template Consumer's template language. The only character set explicitly supported by XML parsers is Unicode; Template Consumers are not required to support any other character set.

Example: XSLT expects the Unicode character set. If the data in the DATA element is encoded in EBCDIC, the Template Consumer must map the incoming EBCDIC characters to the corresponding Unicode characters.

4.7 The <DATA_REF> Element

4.7.1 Description

The DATA_REF element identifies, by reference, a data file that has already been installed in the Template Consumer using a DATA element with the Name and Environment attributes.

4.7.2 Model

```
DATA_REF      Empty
```

4.7.3 Attributes

Attribute	Required /Optional	Type	Description
Ref	Required	String	Name of the previously installed data file. The name must be unique within the environment.
Environment	Required	String	Specifies the environment of the name. (There is no default environment.)
Checksum	Optional	String	Hexadecimal-encoded string, provided as a hint to the Template Consumer as an aid in identifying the data file that was installed earlier. Template 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".

4.7.4 Context

The DATA_REF element occurs only within a PPMLT element.

4.8 The <EXTERNAL_DATA> Element

4.8.1 Description

An EXTERNAL_DATA element identifies, by location and access method, a single content datum (e.g. a template or data file).

The EXTERNAL_DATA type is inherited from the PPML specification, with the addition of the CharacterSet attribute. Any changes to this element in the PPML XML Schema will automatically propagate to the PPMLT schema.

4.8.2 Model

```
EXTERNAL_DATA   EMPTY
```

4.8.3 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.[1] See also application note below.
Checksum	Optional	String	Hexadecimal-encoded string, provided as a hint to the Template Consumer. Template 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".
CharacterSet	Optional	String	Identifies the character set used in the referenced data. Default="UTF-8". See description under INTERNAL_DATA.

4.8.4 Context

EXTERNAL_DATA may occur within TEMPLATE and DATA.

4.8.5 Application note regarding URI

A PPML Template Consumer is not required to support any particular access protocol (for instance, HTTP), so a data emitter cannot be certain that URIs in EXTERNAL_DATA will be readable by an unknown Template Consumer. Therefore, if a data emitter wants to ensure that the template will be readable by any Template Consumer, INTERNAL_DATA should be used.

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

4.9 The <INTERNAL_DATA> Element

4.9.1 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.

4.9.2 Model

```
INTERNAL_DATA    ANY
```

4.9.3 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).[2] However, note that Template 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).[3] Default: the character set of the enclosing PPMLT file.
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.

4.9.4 Context

`INTERNAL_DATA` may occur within `TEMPLATE` and `DATA`.

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

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

4.10 The <DATA_STRUCTURE> Element

4.10.1 Description

The optional DATA_STRUCTURE element describes the structure of the data expected by the PPML Template script.

Note that any method of description is allowed as content of the DATA_STRUCTURE element, e.g. DTD, RELAX, XML Schema or even plain text.

In XML applications, the DATA_STRUCTURE inside TEMPLATE describes the format of the XML expected by that template. This XML may be provided directly or generated at the PPML Template Consumer from non-XML data.

The DATA_STRUCTURE inside DATA describes the structure of that DATA.

4.10.2 Model

```
DATA_STRUCTURE  (INTERNAL_DATA | EXTERNAL_DATA)
```

4.10.3 Attributes

Attribute	Required /Optional	Type	Description
Format	Required	String	The format of this description. Must be a valid MIME type.

4.10.4 Context

The DATA_STRUCTURE element occurs within a TEMPLATE or DATA element.

4.10.5 Notes

It is assumed that most, if not all, initial implementations of PPML Templating will use the DATA_STRUCTURE elements for documentation purposes only. However the information stored there can be used to automate the process of mating databases with templates. See also the DATA_MAPPER element.

4.11 The <DATA_MAPPER> element

4.11.1 Description

The optional DATA_MAPPER element contains a script designed to reformat the input data (specified in the DATA element) to the form expected by a PPML Template script. The result of applying DATA_MAPPER to the DATA becomes the input to TEMPLATE.

If DATA is sent separately from TEMPLATE, each may have an associated DATA_MAPPER element. This allows flexibility in configuring the workflow, for instance to adapt to the needs of individual customers or systems. Examples:

- The incoming data can be transformed into the format expected by the template
- A PPML Template Consumer system could have several DATA_MAPPER scripts available, to accommodate a variety of different incoming data formats.
- Both might be true. For instance, a DATA_MAPPER script may be included in the PPMLT element that includes the DATA element, which transforms the data into the format expected by another DATA_MAPPER script in the PPMLT element that contains the template. In this case, the DATA_MAPPER associated with DATA is applied first.

Optionally, the DATA_MAPPER element may also contain descriptions of the structure of the input and output data describing the input format expected by the script and the output that it will generate. Initial PPML Templating implementations may only use these elements for documentation – to document what format is expected as input to the XSLT script and what format the will output. However future implementations may consider making some intelligent use of the information conveyed in these descriptions.

4.11.2 Model

```
DATA_MAPPER   ((INPUT_DATA_STRUCTURE, OUTPUT_DATA_STRUCTURE)?,
              (INTERNAL_DATA | EXTERNAL_DATA))
```

4.11.3 Attributes

Attribute	Required /Optional	Type	Description
Format	Optional	String	The format of this template. Must be a valid MIME type.
Name	Optional	String	Name to be used when referring to this data mapper. The name must be unique within the environment.
Environment	Optional	String	Specifies the environment in which the mapper's name should be available. (There is no default environment.) Required if the Name attribute is used.

4.11.4 Context

DATA_MAPPER occurs only at the top level, in a PPMLT element.

4.11.5 Notes

If `INPUT_DATA_STRUCTURE` is present, then `OUTPUT_DATA_STRUCTURE` must follow it.

4.11.6 Application note: saving data mappers for later use

When a PPML Template Consumer processes a `DATA_MAPPER` element with a Name attribute (and therefore also with an Environment attribute), the Template Consumer is required to save the mapper for later use, along with the value of the Format attribute.

Management of data mappers installed in a Template Consumer is not specified in this document. The developer of a Template Consumer is responsible for providing a way to manage them, e.g. deleting files that are no longer needed.

4.12 The <DATA_MAPPER_REF> Element

4.12.1 Description

The DATA_MAPPER_REF element identifies, by reference, a data mapper that has already been installed in the Template Consumer using a DATA_MAPPER element with the Name and Environment attributes.

4.12.2 Model

```
DATA_MAPPER_REF  Empty
```

4.12.3 Attributes

Attribute	Required /Optional	Type	Description
Ref	Required	String	Name of the previously installed mapper. The name must be unique within the environment.
Environment	Required	String	Specifies the environment of the name. (There is no default environment.)
Checksum	Optional	String	Hexadecimal-encoded string, provided as a hint to the Template Consumer as an aid in identifying the mapper that was installed earlier. Template 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".

4.12.4 Context

The DATA_REF element occurs only within a PPMLT element.

4.13 The <INPUT_DATA_STRUCTURE> Element

4.13.1 Description

The INPUT_DATA_STRUCTURE element describes the data format of the data being converted by a DATA_MAPPER element. As noted in the previous section, this is provided on an "information only" basis – there is no requirement that the receiving system do anything with this information.

In the case of an XML database, the content of INPUT_DATA_STRUCTURE would be an XML Schema or DTD describing the format of that database.

4.13.2 Model

```
INPUT_DATA_STRUCTURE   (INTERNAL_DATA | EXTERNAL_DATA)
```

4.13.3 Attributes

Attribute	Required /Optional	Type	Description
Format	Required	String	The format of this description.

4.13.4 Context

The INPUT_DATA_STRUCTURE element occurs only within a DATA_MAPPER element.

4.14 The <OUTPUT_DATA_STRUCTURE> Element

4.14.1 Description

Like INPUT_DATA_STRUCTURE, this is an optional "information only" element within DATA_MAPPER. It describes the data format of the output generated by the DATA_MAPPER's script.

4.14.2 Model

```
OUTPUT_DATA_STRUCTURE  (INTERNAL_DATA | EXTERNAL_DATA)
```

4.14.3 Attributes

Attribute	Required /Optional	Type	Description
Format	Required	String	The format of this description.

4.14.4 Context

The OUTPUT_DATA_STRUCTURE element occurs only within a DATA_MAPPER element.

Chapter 5: Data

5.1 Introduction

The most natural data format for input to PPML Templating is XML. However, some data sources (spreadsheets, legacy systems, etc) are not set up to export in XML format; instead, they typically output delimiter-separated values or plain unformatted line data.

This chapter describes an optional but standardized method of encoding record-oriented data into XML, in a PPML Template <R> and <F> structure, which is contained in a root `<RECORDS>` element. This information may be of use for developers of systems who wish to incorporate this ability into their PPML Template Consumer. A pre-processor could also be developed to convert the data to R/F format, placing the result into a separate file which can be referenced using `EXTERNAL_DATA`.

5.1.1 Delimiter-separated values ("DSV")

In this data format, each individual record is on a separate line, and the fields are separated with a delimiter. Typically a comma is used, resulting in the "comma-separated values" format (CSV) that is commonly output by applications such as Microsoft Excel. A common alternative is tab-delimited data.

Converting DSV data to R/F format is trivial. Each line of input data (i.e. each record) becomes an <R> element, and each field value is placed into an <F> element. Example:

CSV data:

```
John,Watson,12 Main St.,Anywhere,NY,10021
Mary,Smith,47 Broadview,OurTown,NH,03079
```

Converted to R/F format:

```
<RECORDS>
    <R>
        <F>John</F>
        <F>Watson</F>
        <F>12 Main St.</F>
        <F>Anywhere</F>
        <F>NY</F>
        <F>10021</F>
    </R>
    <R>
        <F>Mary</F>
        <F>Smith</F>
        <F>47 Broadview</F>
        <F>OurTown</F>
        <F>NH</F>
        <F>03079</F>
    </R>
</RECORDS>
```

5.1.2 Line data

Some computer systems output data with no delimiters. Instead, fields are identified by their fixed column position.

This example shows the same data as in the previous section:

```
          1         2         3         4         5         6         7         8
12345678901234567890123456789012345678901234567890123456789012345678901234567890
John        Watson    12 Main St.       Anywhere      NY     10021
Mary        Smith     47 Broadview      OurTown       NH     03079
```

In this example the text in columns 1-12 is the first field, columns 13-22 is the second field, etc. When converted to R/F format the result would be the same as the example above.

Many application tools are available to parse line data into fields.

5.1.3 Parameterizing these conversions

PPML Template workflow designers may have a choice regarding how to handle data formats that are essentially identical except for certain parameters. For instance tab-delimited data is essentially the same as comma-delimited.

Some workflows may find it more convenient to have separate Mapper scripts (section 4.11) for comma-delimited and tab-delimited cases; these scripts can easily be selected by a reference in `EXTERNAL_DATA`. Others may prefer to design a single script that handles all delimiter-separated files, using a parameter to identify what the delimiter is. In this case the script's parameters could be passed from the Producer to the script via arguments in the URI.

Example: a user might create a universal script for handling delimiter-separated values. The script might accept a parameter named "delim". To use that script for files delimited with tabs (0x09), a URI to reference that script might be (the parameter is shaded for easy identification):

```
<DATA_MAPPER>
  <EXTERNAL_DATA  Src="MyDelimiterScript.xsl?delim=&#x09;">
</DATA_MAPPER>
```

When the same script is used to process a file delimited with commas (0x2C), the URI might be:

```
<DATA_MAPPER>
  <EXTERNAL_DATA  Src="MyDelimiterScript.xsl?delim=&#x2C;">
</DATA_MAPPER>
```

5.2 The <RECORDS> element

5.2.1 Description

The RECORDS element is the top level, providing the root node that contains all the R and F records.

5.2.2 Model

```
RECORDS      (R*)
```

5.2.3 Attributes

None.

5.2.4 Context

Within a PPMLT element, RECORDS occurs only in INTERNAL_DATA. RECORDS may also occur in a separate data file referenced by EXTERNAL_DATA.

5.3 The <R> element

5.3.1 Description

The `R` element contains one record of variable data, consisting of one or more `F` elements.

5.3.2 Model

```
R              (F+)
```

5.3.3 Attributes

None.

5.3.4 Context

`R` occurs only in `DATA`.

5.4 The <F> element

5.4.1 Description

The `F` element contains one field of data within a record.

5.4.2 Models

```
F               (#PCDATA)
```

5.4.3 Attributes

Attribute	Required /Optional	Type	Description
Name	Optional	String	The name of this field. This attribute is provided as a convenience for human readability, including cross-referencing to the original (non-XML) file. It may also be of use in workflows that require named fields.

5.4.4 Context

The `F` element occurs only inside `R`.

5.5 Very long data streams

When presented with very long data streams, such as hundreds of thousands of records, a tree-oriented scripting system such as XSLT can become resource-intensive (causing problems with speed or memory). The Xpath expressions within XSLT have random access to the entire XML data tree, so XSLT processing effectively requires reading the entire XML tree, and most XSLT processors have significant performance problems when the tree is large.

Two alternatives are available to avoid this problem:

- A data emitter can transmit the data in multiple, smaller PPMLT files. See example 1 below.
- PPML Template Consumers are allowed to break up the incoming data into smaller "chunks" at the boundary between record boundaries. For XML data, the Template Consumer is allowed to terminate the tree between immediate children of the root element. See example 2.

  Note: In applications where the template requires multiple records of input data per Instance Document, care must be taken to "chunk" between appropriate record groups. A `DATA_MAPPER` script can be used to pre-process the data for this purpose, as shown in example 3 below.

The following PPMLT file is used in the examples below.

```
<PPMLT>
 <TEMPLATE_REF  Env="Myco"  Ref="MyTemplate"/>
 <DATA>
   <INTERNAL_DATA>
     <RECORDS>
        <R><F>Record1.....</R>
        <R><F>Record2.....</R>
        <R><F>Record3.....</R>
        ....
        <R><F>Record99.....</R>
     </RECORDS>
   </INTERNAL_DATA>
 <DATA>
</PPMLT>
```

Example 1: Data emitter breaks the data into smaller PPMLT files

The data emitter closes off one PPMLT element and opens another one, so that the PPML Template Consumer processes a long job as several small ones. The lines highlighted in blue are inserted:

```
<PPMLT>
 <TEMPLATE_REF  Env="Myco"  Ref="MyTemplate"/>
 <DATA>
   <INTERNAL_DATA>
     <RECORDS>
        <R><F>Record1.....</R>
        <R><F>Record2.....</R>
        ....
        <R><F>Record50.....</R>
     </RECORDS>
   </INTERNAL_DATA>
 <DATA>
</PPMLT>
<PPMLT>
```



```
<TEMPLATE_REF  Env="Myco"  Ref="MyTemplate"/>
<DATA>
  <INTERNAL_DATA>
    <RECORDS>
      <R><F>Record51.....</R>
      ....
      <R><F>Record99.....</R>
    </RECORDS>
  </INTERNAL_DATA>
 <DATA>
</PPMLT>
```

Example 2: Template Consumer "chunks" the data

In this method the Template Consumer breaks the data appropriate boundaries. In XML data, this is at any immediate child of the root element. For the "R/F" record/field structure described in this specification, the root element is `RECORDS` and the immediate children are the `R` records, so the Template Consumer may terminate the tree after any `R` element and start a new tree with the next `R` element, as if another root element had been inserted.

With chunking, the template processor is invoked more than once, and sees different sets of data on the different invocations, for instance:

Invocation 1:

```
<RECORDS>
  <R><F>Record1.....</R>
  <R><F>Record2.....</R>
  <R><F>Record3.....</R>
  ....
  <R><F>Record50.....</R>
</RECORDS>
```

Invocation 2:

```
<RECORDS>
  <R><F>Record51.....</R>
  ....
  <R><F>Record99.....</R>
</RECORDS>
```

Example 3: Using a DATA_MAPPER to perform multi-record grouping

Some templates may require that several records of data be kept together (e.g. applications where the data for one customer occupies three records). In this case, chunking between any immediate child (an R element) could result in invalid grouping of records. This problem can be circumvented by using a Mapper script to group elements together by adding a parent XML element. For instance:

Original ungrouped data:

```
<RECORDS>
  <R><F>Record1.....</R>
  <R><F>Record2.....</R>
  <R><F>Record3.....</R>
  <R><F>Record4.....</R>
  <R><F>Record5.....</R>
  <R><F>Record6.....</R>
  ....
  <R><F>Record97.....</R>
  <R><F>Record98.....</R>
```

```
   <R><F>Record99.....</R>
</RECORDS>
```

After processing by an appropriate Mapper:

In this example, the Template Producer provided a Mapper script that inserts a parent `CUSTOMER` element around sets of three records. ("Customer" is an arbitrary name in this example.) When that Mapper script is executed by the Template Consumer, the above data is transformed into the following structure. The immediate child of `RECORDS` is now `CUSTOMER`, so chunking will break at safe boundaries:

```
<RECORDS>
   <CUSTOMER>
      <R><F>Record1.....</R>
      <R><F>Record2.....</R>
      <R><F>Record3.....</R>
   </CUSTOMER>
   <CUSTOMER>
      <R><F>Record4.....</R>
      <R><F>Record5.....</R>
      <R><F>Record6.....</R>
   </CUSTOMER>
      ...
   <CUSTOMER>
      <R><F>Record97.....</R>
      <R><F>Record98.....</R>
      <R><F>Record99.....</R>
   <CUSTOMER>
</RECORDS>
```

Appendix A: Sample Application

A.1 Introduction

This chapter contains a complete PPML Templating job, with both the `TEMPLATE` and the `DATA` expressed as `INTERNAL_DATA`. It then shows how the job can be made increasingly more efficient by storing more and more repetitive content in the Template Consumer: first the PPML Reusable Object definitions and then the document template.

A.2 Example 1: PPML Templating code, including Reusable Object definitions, complete PPML Template and Data Mapper, and data records

This code totals 14.5k; the 25 records of customer data at the end add 3.5k, for a total of 18k.

```
<?xml version="1.0" encoding="UTF-8"?>
<PPMLT xmlns="http://www.podi.org/ppmlt/ppmlt001.xsd">
   <TEMPLATE Format="application/xslt+xml">
      <INTERNAL_DATA>
        <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
           xmlns:fo="http://www.w3.org/1999/XSL/Format" xmlns:svg="svg" version="1.0">
         <xsl:output indent="yes"/>
         <xsl:strip-space elements="*"/>
         <xsl:template match="/">
            <!-- Copyright Atlas Software BV -->
```

The color-highlighted block below contains the prototype PPML file. The yellow portion will be output once; it contains the start-of-job information, including definitions of reusable content, sheet layout, etc. The blue-shaded portion contains an XSLT "for-each", so it will be output repeatedly, once for each customer record, as explained below.

```
<PPML>
   <DOCUMENT SET Label="Job Number 1">
     <IMPOSITION Name="Imporef">
        <SIGNATURE Nrows="1" Ncols="1">
           <CELL Row="1" Col="1" Face="Up" PageOrder="s"/>
        </SIGNATURE>
     </IMPOSITION>
     <PRINT_LAYOUT>
        <PAGE LAYOUT TrimBox="0 0 612 792"/>
        <SHEET_LAYOUT Hsize="612" Vsize="792">
         <IMPOSITION_REF Name="Imporef"/>
        </SHEET_LAYOUT>
     </PRINT LAYOUT>
     <PRIVATE_INFO Creator="Xeikon" Identifier="MasterVDF"/>
     <REUSABLE OBJECT>
        <OBJECT Position="0 0">
          <SOURCE Format="application/postscript" Dimensions="612 792">
            <EXTERNAL_DATA Src="OldsMobile.eps"/>
          </SOURCE>
        </OBJECT>
        <OCCURRENCE_LIST>
```



```
            <OCCURRENCE Name="XMASTER_ OldsMobile.eps_1 0 0 1 0 0"
               Environment="Demo">
               <VIEW>
                  <TRANSFORM Matrix="1 0 0 1 0 0"/>
               </VIEW>
            </OCCURRENCE>
         </OCCURRENCE_LIST>
      </REUSABLE OBJECT>
      <REUSABLE_OBJECT>
         <OBJECT Position="0 0">
            <SOURCE Format="application/postscript" Dimensions="612 792">
               <INTERNAL_DATA>
                  gsave
                  /psmPaintRect { gsave newpath 4 2 roll moveto 1 index 0
                     rlineto 0
                  exch rlineto neg 0 rlineto fill grestore } bind def
                  0 0 0 0 setcmykcolor
                  0 0 612 792 psmPaintRect
                  grestore
               </INTERNAL_DATA>
            </SOURCE>
         </OBJECT>
         <OCCURRENCE_LIST>
            <OCCURRENCE Name="XMASTER_BackForeground_1_1" Environment="Demo"/>
         </OCCURRENCE LIST>
      </REUSABLE_OBJECT>
      <REUSABLE OBJECT>
         <OBJECT Position="0 0">
            <SOURCE Format="application/postscript" Dimensions="165.4 15">
               <INTERNAL_DATA>
                  gsave
                  /psmPaintRect { gsave newpath 4 2 roll moveto 1 index 0
                     rlineto 0
                  exch rlineto neg 0 rlineto fill grestore } bind def
                  0 0 0 1 setcmykcolor
                  0 0 165.421707153 15.0005950928 psmPaintRect
                  grestore
               </INTERNAL_DATA>
            </SOURCE>
         </OBJECT>
         <OCCURRENCE_LIST>
           <OCCURRENCE Name="BackForeground_2_1" Environment="Demo"/>
         </OCCURRENCE_LIST>
      </REUSABLE OBJECT>
      <REUSABLE_OBJECT>
         <OBJECT Position="0 0">
            <SOURCE Format="application/postscript" Dimensions="165.4 15">
               <INTERNAL_DATA>
                  gsave
                  /psmPaintRect { gsave newpath 4 2 roll moveto 1 index 0
                     rlineto 0
                  exch rlineto neg 0 rlineto fill grestore } bind def
                  0 0 0 1 setcmykcolor
                  0 0 165.421707153 15.0004425049 psmPaintRect
                  grestore
               </INTERNAL_DATA>
            </SOURCE>
         </OBJECT>
         <OCCURRENCE_LIST>
           <OCCURRENCE Name="BackForeground_3_1" Environment="Demo"/>
         </OCCURRENCE_LIST>
      </REUSABLE OBJECT>
      <REUSABLE_OBJECT>
         <OBJECT Position="0 0">
            <SOURCE Format="application/postscript" Dimensions="165.4 14">
               <INTERNAL_DATA>
                  gsave
                  /psmPaintRect { gsave newpath 4 2 roll moveto 1 index 0
                     rlineto 0
                  exch rlineto neg 0 rlineto fill grestore } bind def
```

```
            0 0 0 1 setcmykcolor
            0 0 165.421707153 14.0002288818 psmPaintRect
            grestore
         </INTERNAL_DATA>
      </SOURCE>
   </OBJECT>
   <OCCURRENCE_LIST>
      <OCCURRENCE Name="BackForeground_4_1" Environment="Demo"/>
   </OCCURRENCE_LIST>
</REUSABLE_OBJECT>
<REUSABLE_OBJECT>
   <OBJECT Position="0 0">
      <SOURCE Format="application/postscript" Dimensions="164.4 15">
         <INTERNAL_DATA>
            gsave
            /psmPaintRect { gsave newpath 4 2 roll moveto 1 index 0
               rlineto 0
            exch rlineto neg 0 rlineto fill grestore } bind def
            0 0 0 1 setcmykcolor
            0 0 164.422149658 15.0004425049 psmPaintRect
            grestore
         </INTERNAL_DATA>
      </SOURCE>
   </OBJECT>
   <OCCURRENCE LIST>
      <OCCURRENCE Name="BackForeground_5_1" Environment="Demo"/>
   </OCCURRENCE LIST>
</REUSABLE_OBJECT>
<REUSABLE_OBJECT>
   <OBJECT Position="0 0">
      <SOURCE Format="application/postscript" Dimensions="190.5 12.5">
         <INTERNAL_DATA>
            gsave
            /psmPaintRect { gsave newpath 4 2 roll moveto 1 index 0
               rlineto 0
            exch rlineto neg 0 rlineto fill grestore } bind def
            0 0 0 1 setcmykcolor
            0 0 190.499694824 12.5 psmPaintRect
            grestore
         </INTERNAL_DATA>
      </SOURCE>
   </OBJECT>
   <OCCURRENCE_LIST>
      <OCCURRENCE Name="BackForeground_6_1" Environment="Demo"/>
   </OCCURRENCE_LIST>
</REUSABLE_OBJECT>
<REUSABLE OBJECT>
   <OBJECT Position="0 0">
      <SOURCE Format="application/postscript" Dimensions="191 94">
         <EXTERNAL_DATA Src="PURPLE"/>
      </SOURCE>
   </OBJECT>
   <VIEW>
      <CLIP_RECT Rectangle="0.04066 0.227 191 93.77"/>
   </VIEW>
   <OCCURRENCE_LIST>
      <OCCURRENCE Name="PURPLE_1 0 0 1 -0.04066 -0.227"
         Environment="Demo">
         <VIEW>
            <TRANSFORM Matrix="1 0 0 1 -0.04066 -0.227"/>
         </VIEW>
      </OCCURRENCE>
   </OCCURRENCE_LIST>
</REUSABLE_OBJECT>
<REUSABLE_OBJECT>
   <OBJECT Position="0 0">
     <SOURCE Format="application/postscript" Dimensions="191 94">
        <EXTERNAL_DATA Src="BLUE"/>
     </SOURCE>
   </OBJECT>
```



```
    <VIEW>
      <CLIP_RECT Rectangle="0.04066 0.227 191 93.77"/>
    </VIEW>
    <OCCURRENCE_LIST>
      <OCCURRENCE Name="BLUE_1 0 0 1 -0.04066 -0.227" Environment="Demo">
        <VIEW>
          <TRANSFORM Matrix="1 0 0 1 -0.04066 -0.227"/>
        </VIEW>
      </OCCURRENCE>
    </OCCURRENCE_LIST>
  </REUSABLE_OBJECT>
  <REUSABLE_OBJECT>
    <OBJECT Position="0 0">
      <SOURCE Format="application/postscript" Dimensions="191 94">
        <EXTERNAL_DATA Src="SILVER"/>
      </SOURCE>
    </OBJECT>
    <VIEW>
      <CLIP_RECT Rectangle="0.04066 0.227 191 93.77"/>
    </VIEW>
    <OCCURRENCE LIST>
      <OCCURRENCE Name="SILVER_1 0 0 1 -0.04066 -0.227" Environment="Demo">
      <VIEW>
        <TRANSFORM Matrix="1 0 0 1 -0.04066 -0.227"/>
      </VIEW>
      </OCCURRENCE>
    </OCCURRENCE LIST>
  </REUSABLE_OBJECT>
  <REUSABLE_OBJECT>
    <OBJECT Position="0 0">
      <SOURCE Format="application/postscript" Dimensions="191 94">
        <EXTERNAL_DATA Src="GREENGRAY"/>
      </SOURCE>
    </OBJECT>
    <VIEW>
      <CLIP_RECT Rectangle="0.04066 0.227 191 93.77"/>
    </VIEW>
    <OCCURRENCE LIST>
      <OCCURRENCE Name="GREEN/GRAY_1 0 0 1 -0.04066 -0.227"
          Environment="Demo">
        <VIEW>
          <TRANSFORM Matrix="1 0 0 1 -0.04066 -0.227"/>
        </VIEW>
      </OCCURRENCE>
    </OCCURRENCE_LIST>
  </REUSABLE_OBJECT>
  <REUSABLE OBJECT>
    <OBJECT Position="0 0">
      <SOURCE Format="application/postscript" Dimensions="191 94">
        <EXTERNAL_DATA Src="BLACK"/>
      </SOURCE>
    </OBJECT>
    <VIEW>
      <CLIP_RECT Rectangle="0.04066 0.227 191 93.77"/>
    </VIEW>
    <OCCURRENCE_LIST>
      <OCCURRENCE Name="BLACK_1 0 0 1 -0.04066 -0.227" Environment="Demo">
      <VIEW>
        <TRANSFORM Matrix="1 0 0 1 -0.04066 -0.227"/>
      </VIEW>
      </OCCURRENCE>
    </OCCURRENCE LIST>
  </REUSABLE_OBJECT>
  <REUSABLE_OBJECT>
    <OBJECT Position="0 0">
      <SOURCE Format="application/postscript" Dimensions="191 94">
        <EXTERNAL_DATA Src="GOLD"/>
      </SOURCE>
    </OBJECT>
    <VIEW>
```

```
            <CLIP_RECT Rectangle="0.04066 0.227 191 93.77"/>
         </VIEW>
         <OCCURRENCE_LIST>
            <OCCURRENCE Name="GOLD_1 0 0 1 -0.04066 -0.227" Environment="Demo">
               <VIEW>
                  <TRANSFORM Matrix="1 0 0 1 -0.04066 -0.227"/>
               </VIEW>
            </OCCURRENCE>
         </OCCURRENCE_LIST>
      </REUSABLE_OBJECT>
      <REUSABLE_OBJECT>
         <OBJECT Position="0 0">
           <SOURCE Format="application/postscript" Dimensions="191 94">
              <EXTERNAL_DATA Src="RED"/>
           </SOURCE>
         </OBJECT>
         <VIEW>
           <CLIP_RECT Rectangle="0.04066 0.227 191 93.77"/>
         </VIEW>
         <OCCURRENCE_LIST>
           <OCCURRENCE Name="RED_1 0 0 1 -0.04066 -0.227" Environment="Demo">
              <VIEW>
                 <TRANSFORM Matrix="1 0 0 1 -0.04066 -0.227"/>
              </VIEW>
           </OCCURRENCE>
         </OCCURRENCE_LIST>
      </REUSABLE OBJECT>
      <REUSABLE_OBJECT>
         <OBJECT Position="0 0">
            <SOURCE Format="application/postscript" Dimensions="190.9 93.55">
               <INTERNAL_DATA>
                  gsave
                  /psmPaintRect { gsave newpath 4 2 roll moveto 1 index 0
                     rlineto 0
                  exch rlineto neg 0 rlineto fill grestore } bind def
                  0 0 0 0 setcmykcolor
                  0 0 190.918685913 93.5459136963 psmPaintRect
                  grestore
               </INTERNAL_DATA>
            </SOURCE>
         </OBJECT>
         <OCCURRENCE LIST>
            <OCCURRENCE Name="BackForeground_7_1" Environment="Demo"/>
         </OCCURRENCE LIST>
      </REUSABLE_OBJECT>
```

The following blue-shaded copy will be output once for each XML `CUSTOMER` *element. In PPML without templating, the blue portion (approximately 4600 bytes) would be output once for each customer record.*

```
<xsl:for-each select="//CUSTOMER">
   <DOCUMENT>
      <PAGE>
         <MARK Position="0 0">
            <OCCURRENCE_REF Ref="XMASTER_BackForeground_1_1"
               Environment="Demo"/>
         </MARK>
         <MARK Position="0 0">
            <OCCURRENCE REF Ref="XMASTER_ OldsMobile.eps_1 0 0 1 0 0"
               Environment="Demo"/>
         </MARK>
         <MARK Position="334 605">
            <OCCURRENCE_REF Ref="BackForeground_2_1" Environment="Demo"/>
         </MARK>
         <MARK Position="334 605">
            <OBJECT Position="0 0">
               <SOURCE Format="image/svg-xml" Dimensions="165 15">
                 <INTERNAL_DATA>
                 <svg:svg width="165pt" height="15pt">
```

```
                <svg:text x="82.5pt" y="10pt" font-family="Helvetica" font-
                    size="10pt" word-spacing="1.294pt" letter-spacing=".129pt"
                    text-anchor="middle" fill="rgb(255,255,255)">
                    <xsl:value-of select="NAME"/>
                </svg:text>
            </svg:svg>
            </INTERNAL_DATA>
        </SOURCE>
    <VIEW>
        <TRANSFORM Matrix="1 0 0 1 0 0"/>
    </VIEW>
</OBJECT>
</MARK>
<MARK Position="334 548.5">
    <OCCURRENCE_REF Ref="BackForeground_3_1" Environment="Demo"/>
</MARK>
<MARK Position="334 548.5">
    <OBJECT Position="0 0">
        <SOURCE Format="image/svg-xml " Dimensions="165 15">
            <INTERNAL_DATA>
                <svg:svg width="165pt" height="15pt">
                    <svg:text x="82.5pt" y="10pt" font-family="Helvetica"
                        font-size="10pt" word-spacing="1.021pt" letter-
                        spacing="0.102pt" text-anchor="middle"
                        fill="rgb(255,255,255)">
                        <xsl:value-of select="STREET"/>
                    </svg:text>
                </svg:svg>
            </INTERNAL_DATA>
        </SOURCE>
        <VIEW>
            <TRANSFORM Matrix="1 0 0 1 0 0"/>
        </VIEW>
    </OBJECT>
</MARK>
<MARK Position="334 494">
    <OCCURRENCE_REF Ref="BackForeground_4_1" Environment="Demo"/>
</MARK>
<MARK Position="334 494">
    <OBJECT Position="0 0">
        <SOURCE Format="image/svg-xml " Dimensions="165 14">
            <INTERNAL DATA>
                <svg:svg width="165pt" height="14pt">
                    <svg:text x="82.5pt" y="9.5pt" font-family="Helvetica"
                        font-size="10pt" letter-spacing=".341pt" text-
                        anchor="middle" fill="rgb(255,255,255)">
                        <xsl:value-of select="PHONE"/>
                    </svg:text>
                </svg:svg>
            </INTERNAL_DATA>
        </SOURCE>
        <VIEW>
            <TRANSFORM Matrix="1 0 0 1 0 0"/>
        </VIEW>
    </OBJECT>
</MARK>
<MARK Position="334 438">
    <OCCURRENCE_REF Ref="BackForeground_5_1" Environment="Demo"/>
</MARK>
<MARK Position="334 438">
    <OBJECT Position="0 0">
        <SOURCE Format="image/svg-xml " Dimensions="164 15">
            <INTERNAL_DATA>
                <svg:svg width="164pt" height="15pt">
                    <svg:text x="82pt" y="9.75pt" font-family="Helvetica"
                        font-size="9pt" letter-spacing="0.05pt" text-
                        anchor="middle" fill="rgb(255,255,255)">
                        <xsl:value-of select="EMAIL"/>
                    </svg:text>
                </svg:svg>
```

```
                </INTERNAL_DATA>
            </SOURCE>
            <VIEW>
                <TRANSFORM Matrix="1 0 0 1 0 0"/>
            </VIEW>
        </OBJECT>
    </MARK>
    <MARK Position="84 582.5">
        <OCCURRENCE_REF Ref="BackForeground_6_1" Environment="Demo"/>
    </MARK>
    <MARK Position="84 582.5">
        <OBJECT Position="0 0">
            <SOURCE Format="image/svg-xml " Dimensions="190 13">
              <INTERNAL DATA>
                <svg:svg width="190pt" height="13pt">
                  <svg:text x="95pt" y="8.5pt" font-family="Helvetica" font-
                      size="8pt" word-spacing="1.789pt" letter-
                      spacing="0.179pt" text-anchor="middle"
                      fill="rgb(255,255,255)">
                      <xsl:value-of select="DESCRIPTION"/>
                  </svg:text>
                </svg:svg>
              </INTERNAL_DATA>
            </SOURCE>
            <VIEW>
                <TRANSFORM Matrix="1 0 0 1 0 0"/>
            </VIEW>
        </OBJECT>
    </MARK>
    <MARK Position="84 598.2">
        <OCCURRENCE_REF Ref="BackForeground_7_1" Environment="Demo"/>
    </MARK>
    <MARK Position="84 598.2">
        <OCCURRENCE_REF Environment="Demo">
            <xsl:attribute name="Ref">
            <xsl:value-of select="IMAGE"/>
            <xsl:text>_1 0 0 1 -0.04066 -0.227</xsl:text>
            </xsl:attribute>
        </OCCURRENCE_REF>
    </MARK>
  </PAGE>
 </DOCUMENT>
</xsl:for-each>
```

Note for comparison: In PPML without templating, each record would add another copy of the `DOCUMENT` *element shown in blue above. This would lengthen the file by about 4.5k per record. In this example with 25 records, the PPML code would be 122k longer, for a total of approximately 140k. If it were fully shown in this specification, this example without templating would be 56 pages long.*

```
        </DOCUMENT_SET>
      </PPML>
    </xsl:template>
  </xsl:stylesheet>
 </INTERNAL_DATA>
</TEMPLATE>
```

The data mapper element is the same in the first two versions of this example.

```
<DATA_MAPPER Format="application/xslt+xml">
  <INTERNAL_DATA>
    <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
      xmlns:fo="http://www.w3.org/1999/XSL/Format">
      <xsl:output indent="yes"/>
      <xsl:template match="R">
        <CUSTOMER>
          <NAME>
            <xsl:value-of select="F[1]"/>
          </NAME>
```

```
            <STREET>
              <xsl:value-of select="F[2]"/>
            </STREET>
            <PHONE>
              <xsl:value-of select="F[3]"/>
            </PHONE>
            <EMAIL>
              <xsl:value-of select="F[4]"/>
            </EMAIL>
            <DESCRIPTION>
              <xsl:value-of select="F[5]"/>
            </DESCRIPTION>
            <IMAGE>
              <xsl:value-of select="F[6]"/>
            </IMAGE>
          </CUSTOMER>
        </xsl:template>
        <xsl:template match="/">
          <CUSTOMERS>
            <xsl:apply-templates/>
          </CUSTOMERS>
        </xsl:template>
      </xsl:stylesheet>
    </INTERNAL_DATA>
</DATA_MAPPER>
```

The Data element is the same in all versions of this example.

```
<DATA Format="application/xml">
  <INTERNAL_DATA>
      <RECORDS>
        <R><F>Cynthia Proctor</F><F>625 Missouri Street</F><F>510-372-7500</F>
          <F>dcgraphicdesigns@hotmail.com</F><F>1998 Purple Intrigue</F>
          <F>PURPLE</F></R>
        <R><F>Dr. Loose</F><F>Whoville</F><F>123-345-5678</F>
          <F>Loose@whoville.com</F><F>1998 Blue Intrigue</F><F>BLUE</F></R>
        <R><F>Henry Polard</F><F>33 World Trade Blvd.</F><F>650-855-9367</F>
          <F>polard@wenet.net</F><F>1998 Silver Intrigue</F><F>SILVER</F></R>
        <R><F>Al Joshua</F><F>4567 My Way</F><F>123-456-789</F>
          <F>ajoshua@psmail.com</F><F>1998 Silver Intrigue</F><F>SILVER</F></R>
        <R><F>Michelle Walker</F><F>860 36th Ave.</F><F>415/831-1019</F>
          <F>shelwalker@aol.com</F><F>1998 Green/gray Intrigue</F>
          <F>GREEN/GRAY</F></R>
        <R><F>Craig Kohler</F><F>860 36th Ave.</F><F>415/831/1019</F>
          <F>craig.kohler@schwab.com</F><F>1998 Black Intrigue</F><F>BLACK</F></R>
        <R><F>Ken Griffith</F><F>34286 Quartz St.</F><F>510-796-4975</F>
          <F>ken_griffith@splashtech.com</F><F>1998 White Intrigue</F>
          <F>WHITE</F></R>
        <R><F>Harry Raaphorst</F><F>Buys Ballotstraat 17-19</F><F>31-341-426700</F>
          <F>harry.raaphorst@atlasssoftware.nl</F><F>1998 Blue
          Intrigue</F><F>BLUE</F></R>
        <R><F>Michael Barnes</F><F>The Maxwell Company</F><F>415-123-4567</F>
          <F>maxwell@edu</F><F>1998 Gold Intrigue</F><F>GOLD</F></R>
        <R><F>Gregg Fox</F><F>200 Canal View Blvd. 831</F><F>716-427-4262</F>
          <F>gregg_fox@mc.xerox.com</F><F>1998 Gold Intrigue</F><F>GOLD</F></R>
        <R><F>Paul Lorton, Jr</F><F>1265 Altschul Av.</F><F>650-854-
          2406</F><F>lorton@usfca.edu</F><F>1998 Red Intrigue</F><F>RED</F></R>
        <R><F>Linda Jackson</F><F>405 - 1263 Barclay Street</F><F>604-844-2253</F>
          <F>linda_jackson@splashtech.com</F><F>1998 Black Intrigue</F> <F>BLACK</F>
          </R>
        <R><F>Denis Severson</F><F>3400 Hillview Ave.</F><F>650-813-7158</F>
          <F>severson@parc.xerox.com</F><F>1998 Red Intrigue</F><F>RED</F></R>
        <R><F>Jindong Chen</F><F>3400 Hillview Ave, PAHV 12</F><F>650-813-7338</F>
          <F>jchen@parc.xerox.com</F><F>1998 Gold Intrigue</F><F>GOLD</F></R>
        <R><F>Gary Roth</F><F>8758 Wescott Court</F><F>619-484-3226</F>
          <F>gary_roth@splashtech.com</F><F>1998 Blue Intrigue</F><F>BLUE</F></R>
        <R><F>Susan Prischmann</F><F>3930 North Pinegrove, Apt.</F><F>312-849-4361</F>
          <F>sprischmann@currentassets.com</F><F>1998 Green/charcoal
          Intrigue</F><F>GREENCHARCOAL</F></R>
```



```
            <R><F>Sue Hoffmann</F><F>2000 Powell Street</F><F>657-1777</F>
               <F>sue_hoffmann@thenet.com</F> <F>1998 Red Intrigue</F><F>RED</F></R>
            <R><F>Rick Placak</F><F>130 So. Center</F><F>702-329-
               3145</F><F>rplacak@thenet.com</F><F>1998 Red Intrigue</F><F>RED</F></R>
            <R><F>Betsy Pryser</F><F>1130 N. Dearborn, #1603</F><F>312-397-
               9250</F><F>epryser@ix.netcom.com</F><F>1998 Silver
               Intrigue</F><F>SILVER</F></R>
            <R><F>Mike Mayo</F><F>124 West Oxmoor</F><F>205-942-
               2222</F><F>jmmayo@worldnet.att.net</F><F>1998 Gold
               Intrigue</F><F>GOLD</F></R>
            <R><F>Armand Petri</F><F>1508 Blackhawk Drive</F><F>408 735
               9482</F><F>apetri@aol.com</F><F>1998 Purple Intrigue</F><F>PURPLE</F></R>
            <R><F>Ted DiSilvestre</F><F>333 W. San Carlos St.</F><F>408-536-
               6508</F><F>tdisilve@adobe.com</F><F>1998 Blue Intrigue</F><F>BLUE</F></R>
            <R><F>Dean Griswold</F><F>6947 West Oak Ct.</F><F>916-725-7739</F>
               <F>griswold@ix.netcom.com</F><F>1998 Green/gray Intrigue</F>
               <F>GREEN/GRAY</F></R>
            <R><F>John Doe</F><F>46 Nowhere Street</F><F>654-321-0987</F>
               <F>john_doe@nowhere.com</F><F>1998 Black Intrigue</F><F>BLACK</F></R>
            <R><F>Jenny Jones</F><F>69 Talkshow Road</F><F>543-210-
               9876</F><F>jjones@tv.com</F><F>1998 Red Intrigue</F><F>RED</F></R>
         </RECORDS>
      </INTERNAL_DATA>
   </DATA>
</PPMLT>
```

A.3 The same dataset, if the Reusable Object occurrences were defined and downloaded earlier

For recurring print projects, a central benefit of PPML is its ability to reference Reusable Object content that was defined earlier using Global scope. This feature allows transmitting smaller datasets and eliminates redundant processing of the reusable content.

The following code shows the PPML Templating dataset that produces the same output as above, presuming that the reusable content was downloaded earlier. The original version was approximately 18k of XML; this version is 10k.

```
<?xml version="1.0" encoding="UTF-8"?>
<PPMLT xmlns="http://www.podi.org/ppmlt/ppmlt001.xsd">
   <TEMPLATE Format="application/xslt+xml">
      <INTERNAL_DATA>
         <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
      xmlns:fo="http://www.w3.org/1999/XSL/Format" xmlns:svg="svg" version="1.0">
            <xsl:output indent="yes"/>
            <xsl:strip-space elements="*"/>
            <xsl:template match="/">
               <!-- Copyright Atlas Software BV -->
```

The start-of-file information is much shorter.

```
            <PPML>
               <DOCUMENT_SET Label="Job Number 1">
                  <IMPOSITION Name="Imporef">
                     <SIGNATURE Nrows="1" Ncols="1">
                        <CELL Row="1" Col="1" Face="Up" PageOrder="s"/>
                     </SIGNATURE>
                  </IMPOSITION>
                  <PRINT_LAYOUT>
                     <PAGE_LAYOUT TrimBox="0 0 612 792"/>
                     <SHEET LAYOUT Hsize="612" Vsize="792">
                        <IMPOSITION_REF Name="Imporef"/>
                     </SHEET LAYOUT>
                  </PRINT_LAYOUT>
                  <PRIVATE_INFO Creator="Xeikon" Identifier="MasterVDF"/>
```

The blue-shaded portion, which defines the PPML Document element that will be output for each `CUSTOMER` *element, is the same as shown above.*

```
<xsl:for-each select="//CUSTOMER">
    <DOCUMENT>
      <PAGE>
        <MARK Position="0 0">
          <OCCURRENCE_REF Ref="XMASTER_BackForeground_1_1"
            Environment="Demo"/>
        </MARK>
        <MARK Position="0 0">
          <OCCURRENCE_REF Ref="XMASTER_ OldsMobile.eps_1 0 0 1 0 0"
            Environment="Demo"/>
        </MARK>
        <MARK Position="334 605">
          <OCCURRENCE_REF Ref="BackForeground_2_1" Environment="Demo"/>
        </MARK>
        <MARK Position="334 605">
          <OBJECT Position="0 0">
            <SOURCE Format="image/svg-xml" Dimensions="165 15">
              <INTERNAL_DATA>
              <svg:svg width="165pt" height="15pt">
                <svg:text x="82.5pt" y="10pt" font-family="Helvetica" font-
                  size="10pt" word-spacing="1.294pt" letter-spacing=".129pt"
                  text-anchor="middle" fill="rgb(255,255,255)">
                  <xsl:value-of select="NAME"/>
                </svg:text>
              </svg:svg>
              </INTERNAL_DATA>
            </SOURCE>
          <VIEW>
            <TRANSFORM Matrix="1 0 0 1 0 0"/>
          </VIEW>
        </OBJECT>
        </MARK>
        <MARK Position="334 548.5">
          <OCCURRENCE_REF Ref="BackForeground_3_1" Environment="Demo"/>
        </MARK>
        <MARK Position="334 548.5">
          <OBJECT Position="0 0">
            <SOURCE Format="image/svg-xml " Dimensions="165 15">
              <INTERNAL_DATA>
                <svg:svg width="165pt" height="15pt">
                  <svg:text x="82.5pt" y="10pt" font-family="Helvetica"
                    font-size="10pt" word-spacing="1.021pt" letter-
                    spacing="0.102pt" text-anchor="middle"
                    fill="rgb(255,255,255)">
                    <xsl:value-of select="STREET"/>
                  </svg:text>
                </svg:svg>
              </INTERNAL_DATA>
            </SOURCE>
            <VIEW>
              <TRANSFORM Matrix="1 0 0 1 0 0"/>
            </VIEW>
          </OBJECT>
        </MARK>
        <MARK Position="334 494">
          <OCCURRENCE_REF Ref="BackForeground_4_1" Environment="Demo"/>
        </MARK>
        <MARK Position="334 494">
          <OBJECT Position="0 0">
            <SOURCE Format="image/svg-xml " Dimensions="165 14">
              <INTERNAL DATA>
                <svg:svg width="165pt" height="14pt">
                  <svg:text x="82.5pt" y="9.5pt" font-family="Helvetica"
                    font-size="10pt" letter-spacing=".341pt" text-
                    anchor="middle" fill="rgb(255,255,255)">
                    <xsl:value-of select="PHONE"/>
```

```
                    </svg:text>
                </svg:svg>
            </INTERNAL_DATA>
        </SOURCE>
        <VIEW>
            <TRANSFORM Matrix="1 0 0 1 0 0"/>
        </VIEW>
    </OBJECT>
</MARK>
<MARK Position="334 438">
    <OCCURRENCE_REF Ref="BackForeground_5_1" Environment="Demo"/>
</MARK>
<MARK Position="334 438">
    <OBJECT Position="0 0">
        <SOURCE Format="image/svg-xml " Dimensions="164 15">
            <INTERNAL_DATA>
                <svg:svg width="164pt" height="15pt">
                    <svg:text x="82pt" y="9.75pt" font-family="Helvetica"
                        font-size="9pt" letter-spacing="0.05pt" text-
                        anchor="middle" fill="rgb(255,255,255)">
                        <xsl:value-of select="EMAIL"/>
                    </svg:text>
                </svg:svg>
            </INTERNAL_DATA>
        </SOURCE>
        <VIEW>
            <TRANSFORM Matrix="1 0 0 1 0 0"/>
        </VIEW>
    </OBJECT>
</MARK>
<MARK Position="84 582.5">
    <OCCURRENCE_REF Ref="BackForeground_6_1" Environment="Demo"/>
</MARK>
<MARK Position="84 582.5">
    <OBJECT Position="0 0">
        <SOURCE Format="image/svg-xml " Dimensions="190 13">
            <INTERNAL_DATA>
                <svg:svg width="190pt" height="13pt">
                    <svg:text x="95pt" y="8.5pt" font-family="Helvetica" font-
                        size="8pt" word-spacing="1.789pt" letter-
                        spacing="0.179pt" text-anchor="middle"
                        fill="rgb(255,255,255)">
                        <xsl:value-of select="DESCRIPTION"/>
                    </svg:text>
                </svg:svg>
            </INTERNAL_DATA>
        </SOURCE>
        <VIEW>
            <TRANSFORM Matrix="1 0 0 1 0 0"/>
        </VIEW>
    </OBJECT>
</MARK>
<MARK Position="84 598.2">
    <OCCURRENCE_REF Ref="BackForeground_7_1" Environment="Demo"/>
</MARK>
<MARK Position="84 598.2">
    <OCCURRENCE_REF Environment="Demo">
        <xsl:attribute name="Ref">
        <xsl:value-of select="IMAGE"/>
        <xsl:text>_1 0 0 1 -0.04066 -0.227</xsl:text>
        </xsl:attribute>
    </OCCURRENCE_REF>
</MARK>
</PAGE>
</DOCUMENT>
</xsl:for-each>
</DOCUMENT_SET>
</PPML>
</xsl:template>
</xsl:stylesheet>
```

```
   </INTERNAL_DATA>
</TEMPLATE>
```

The Data Mapper script (shown here in gray) and the Data element are the same as shown above.

```
<DATA_MAPPER Format="application/xslt+xml">
   <INTERNAL_DATA>
      <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
         xmlns:fo="http://www.w3.org/1999/XSL/Format">
        <xsl:output indent="yes"/>
        <xsl:template match="R">
           <CUSTOMER>
              <NAME>
                <xsl:value-of select="F[1]"/>
              </NAME>
              <STREET>
                <xsl:value-of select="F[2]"/>
              </STREET>
              <PHONE>
                <xsl:value-of select="F[3]"/>
              </PHONE>
              <EMAIL>
                <xsl:value-of select="F[4]"/>
              </EMAIL>
              <DESCRIPTION>
                <xsl:value-of select="F[5]"/>
              </DESCRIPTION>
              <IMAGE>
                <xsl:value-of select="F[6]"/>
              </IMAGE>
           </CUSTOMER>
        </xsl:template>
        <xsl:template match="/">
           <CUSTOMERS>
              <xsl:apply-templates/>
           </CUSTOMERS>
        </xsl:template>
      </xsl:stylesheet>
   </INTERNAL_DATA>
</DATA_MAPPER>
<DATA Format="application/xml">
  <INTERNAL_DATA>
      <RECORDS>
         <R><F>Cynthia Proctor</F><F>625 Missouri Street</F><F>510-372-7500</F>
            <F>dcgraphicdesigns@hotmail.com</F><F>1998 Purple Intrigue</F>
            <F>PURPLE</F></R>
         <R><F>Dr. Loose</F><F>Whoville</F><F>123-345-5678</F>
            <F>Loose@whoville.com</F><F>1998 Blue Intrigue</F><F>BLUE</F></R>
         <R><F>Henry Polard</F><F>33 World Trade Blvd.</F><F>650-855-9367</F>
            <F>polard@wenet.net</F><F>1998 Silver Intrigue</F><F>SILVER</F></R>
         <R><F>Al Joshua</F><F>4567 My Way</F><F>123-456-789</F>
            <F>ajoshua@psmail.com</F><F>1998 Silver Intrigue</F><F>SILVER</F></R>
         <R><F>Michelle Walker</F><F>860 36th Ave.</F><F>415/831-1019</F>
            <F>shelwalker@aol.com</F><F>1998 Green/gray Intrigue</F>
            <F>GREEN/GRAY</F></R>
         <R><F>Craig Kohler</F><F>860 36th Ave.</F><F>415/831/1019</F>
            <F>craig.kohler@schwab.com</F><F>1998 Black Intrigue</F><F>BLACK</F></R>
         <R><F>Ken Griffith</F><F>34286 Quartz St.</F><F>510-796-4975</F>
            <F>ken_griffith@splashtech.com</F><F>1998 White Intrigue</F>
            <F>WHITE</F></R>
         <R><F>Harry Raaphorst</F><F>Buys Ballotstraat 17-19</F><F>31-341-426700</F>
            <F>harry.raaphorst@atlasssoftware.nl</F><F>1998 Blue
            Intrigue</F><F>BLUE</F></R>
         <R><F>Michael Barnes</F><F>The Maxwell Company</F><F>415-123-4567</F>
            <F>maxwell@edu</F><F>1998 Gold Intrigue</F><F>GOLD</F></R>
         <R><F>Gregg Fox</F><F>200 Canal View Blvd. 831</F><F>716-427-4262</F>
            <F>gregg_fox@mc.xerox.com</F><F>1998 Gold Intrigue</F><F>GOLD</F></R>
         <R><F>Paul Lorton, Jr</F><F>1265 Altschul Av.</F><F>650-854-
            2406</F><F>lorton@usfca.edu</F><F>1998 Red Intrigue</F><F>RED</F></R>
```



```
            <R><F>Linda Jackson</F><F>405 - 1263 Barclay Street</F><F>604-844-2253</F>
               <F>linda_jackson@splashtech.com</F><F>1998 Black Intrigue</F> <F>BLACK</F>
               </R>
            <R><F>Denis Severson</F><F>3400 Hillview Ave.</F><F>650-813-7158</F>
               <F>severson@parc.xerox.com</F><F>1998 Red Intrigue</F><F>RED</F></R>
            <R><F>Jindong Chen</F><F>3400 Hillview Ave, PAHV 12</F><F>650-813-7338</F>
               <F>jchen@parc.xerox.com</F><F>1998 Gold Intrigue</F><F>GOLD</F></R>
            <R><F>Gary Roth</F><F>8758 Wescott Court</F><F>619-484-3226</F>
               <F>gary_roth@splashtech.com</F><F>1998 Blue Intrigue</F><F>BLUE</F></R>
            <R><F>Susan Prischmann</F><F>3930 North Pinegrove, Apt.</F><F>312-849-4361</F>
               <F>sprischmann@currentassets.com</F><F>1998 Green/charcoal
               Intrigue</F><F>GREENCHARCOAL</F></R>
            <R><F>Sue Hoffmann</F><F>2000 Powell Street</F><F>657-1777</F>
               <F>sue_hoffmann@thenet.com</F> <F>1998 Red Intrigue</F><F>RED</F></R>
            <R><F>Rick Placak</F><F>130 So. Center</F><F>702-329-
               3145</F><F>rplacak@thenet.com</F><F>1998 Red Intrigue</F><F>RED</F></R>
            <R><F>Betsy Pryser</F><F>1130 N. Dearborn, #1603</F><F>312-397-
               9250</F><F>epryser@ix.netcom.com</F><F>1998 Silver
               Intrigue</F><F>SILVER</F></R>
            <R><F>Mike Mayo</F><F>124 West Oxmoor</F><F>205-942-
               2222</F><F>jmmayo@worldnet.att.net</F><F>1998 Gold
               Intrigue</F><F>GOLD</F></R>
            <R><F>Armand Petri</F><F>1508 Blackhawk Drive</F><F>408 735
               9482</F><F>apetri@aol.com</F><F>1998 Purple Intrigue</F><F>PURPLE</F></R>
            <R><F>Ted DiSilvestre</F><F>333 W. San Carlos St.</F><F>408-536-
               6508</F><F>tdisilve@adobe.com</F><F>1998 Blue Intrigue</F><F>BLUE</F></R>
            <R><F>Dean Griswold</F><F>6947 West Oak Ct.</F><F>916-725-7739</F>
               <F>griswold@ix.netcom.com</F><F>1998 Green/gray Intrigue</F>
               <F>GREEN/GRAY</F></R>
            <R><F>John Doe</F><F>46 Nowhere Street</F><F>654-321-0987</F>
               <F>john_doe@nowhere.com</F><F>1998 Black Intrigue</F><F>BLACK</F></R>
            <R><F>Jenny Jones</F><F>69 Talkshow Road</F><F>543-210-
               9876</F><F>jjones@tv.com</F><F>1998 Red Intrigue</F><F>RED</F></R>
         </RECORDS>
      </INTERNAL_DATA>
   </DATA>
</PPMLT>
```

A.4 Leanest form: Template, Reusable Content, and Data Mapper have all been downloaded in advance

In this version the XML code is reduced to 3.8k – essentially the size of the variable data itself. The TEMPLATE and DATA_MAPPER elements each contain an EXTERNAL_DATA reference to a previously defined file that was downloaded earlier. Similar results could have been achieved using TEMPLATE_REF and DATA_MAPPER_REF.

```
<?xml version="1.0" encoding="UTF-8"?>
<PPMLT xmlns="http://www.podi.org/ppmlt/ppmlt001.xsd">
   <TEMPLATE Format="application/xslt+xml">
      <EXTERNAL_DATA Src="Project.xsl"/>
   </TEMPLATE>
   <DATA_MAPPER Format="application/xslt+xml">
      <EXTERNAL_DATA  Src="MyMapper.xsl"/>
   </DATA_MAPPER>
   <DATA Format="application/xml">
     <INTERNAL_DATA>
        <RECORDS>
           <R><F>Cynthia Proctor</F><F>625 Missouri Street</F><F>510-372-7500</F>
              <F>dcgraphicdesigns@hotmail.com</F><F>1998 Purple Intrigue</F>
              <F>PURPLE</F></R>
           <R><F>Dr. Loose</F><F>Whoville</F><F>123-345-5678</F>
              <F>Loose@whoville.com</F><F>1998 Blue Intrigue</F><F>BLUE</F></R>
           <R><F>Henry Polard</F><F>33 World Trade Blvd.</F><F>650-855-9367</F>
              <F>polard@wenet.net</F><F>1998 Silver Intrigue</F><F>SILVER</F></R>
           <R><F>Al Joshua</F><F>4567 My Way</F><F>123-456-789</F>
              <F>ajoshua@psmail.com</F><F>1998 Silver Intrigue</F><F>SILVER</F></R>
```

```
            <R><F>Michelle Walker</F><F>860 36th Ave.</F><F>415/831-1019</F>
               <F>shelwalker@aol.com</F><F>1998 Green/gray Intrigue</F>
               <F>GREEN/GRAY</F></R>
            <R><F>Craig Kohler</F><F>860 36th Ave.</F><F>415/831/1019</F>
               <F>craig.kohler@schwab.com</F><F>1998 Black Intrigue</F><F>BLACK</F></R>
            <R><F>Ken Griffith</F><F>34286 Quartz St.</F><F>510-796-4975</F>
               <F>ken_griffith@splashtech.com</F><F>1998 White Intrigue</F>
               <F>WHITE</F></R>
            <R><F>Harry Raaphorst</F><F>Buys Ballotstraat 17-19</F><F>31-341-426700</F>
               <F>harry.raaphorst@atlasssoftware.nl</F><F>1998 Blue
               Intrigue</F><F>BLUE</F></R>
            <R><F>Michael Barnes</F><F>The Maxwell Company</F><F>415-123-4567</F>
               <F>maxwell@edu</F><F>1998 Gold Intrigue</F><F>GOLD</F></R>
            <R><F>Gregg Fox</F><F>200 Canal View Blvd. 831</F><F>716-427-4262</F>
               <F>gregg_fox@mc.xerox.com</F><F>1998 Gold Intrigue</F><F>GOLD</F></R>
            <R><F>Paul Lorton, Jr</F><F>1265 Altschul Av.</F><F>650-854-
               2406</F><F>lorton@usfca.edu</F><F>1998 Red Intrigue</F><F>RED</F></R>
            <R><F>Linda Jackson</F><F>405 - 1263 Barclay Street</F><F>604-844-2253</F>
               <F>linda_jackson@splashtech.com</F><F>1998 Black Intrigue</F> <F>BLACK</F>
               </R>
            <R><F>Denis Severson</F><F>3400 Hillview Ave.</F><F>650-813-7158</F>
               <F>severson@parc.xerox.com</F><F>1998 Red Intrigue</F><F>RED</F></R>
            <R><F>Jindong Chen</F><F>3400 Hillview Ave, PAHV 12</F><F>650-813-7338</F>
               <F>jchen@parc.xerox.com</F><F>1998 Gold Intrigue</F><F>GOLD</F></R>
            <R><F>Gary Roth</F><F>8758 Wescott Court</F><F>619-484-3226</F>
               <F>gary_roth@splashtech.com</F><F>1998 Blue Intrigue</F><F>BLUE</F></R>
            <R><F>Susan Prischmann</F><F>3930 North Pinegrove, Apt.</F><F>312-849-4361</F>
               <F>sprischmann@currentassets.com</F><F>1998 Green/charcoal
               Intrigue</F><F>GREENCHARCOAL</F></R>
            <R><F>Sue Hoffmann</F><F>2000 Powell Street</F><F>657-1777</F>
               <F>sue_hoffmann@thenet.com</F> <F>1998 Red Intrigue</F><F>RED</F></R>
            <R><F>Rick Placak</F><F>130 So. Center</F><F>702-329-
               3145</F><F>rplacak@thenet.com</F><F>1998 Red Intrigue</F><F>RED</F></R>
            <R><F>Betsy Pryser</F><F>1130 N. Dearborn, #1603</F><F>312-397-
               9250</F><F>epryser@ix.netcom.com</F><F>1998 Silver
               Intrigue</F><F>SILVER</F></R>
            <R><F>Mike Mayo</F><F>124 West Oxmoor</F><F>205-942-
               2222</F><F>jmmayo@worldnet.att.net</F><F>1998 Gold
               Intrigue</F><F>GOLD</F></R>
            <R><F>Armand Petri</F><F>1508 Blackhawk Drive</F><F>408 735
               9482</F><F>apetri@aol.com</F><F>1998 Purple Intrigue</F><F>PURPLE</F></R>
            <R><F>Ted DiSilvestre</F><F>333 W. San Carlos St.</F><F>408-536-
               6508</F><F>tdisilve@adobe.com</F><F>1998 Blue Intrigue</F><F>BLUE</F></R>
            <R><F>Dean Griswold</F><F>6947 West Oak Ct.</F><F>916-725-7739</F>
               <F>griswold@ix.netcom.com</F><F>1998 Green/gray Intrigue</F>
               <F>GREEN/GRAY</F></R>
            <R><F>John Doe</F><F>46 Nowhere Street</F><F>654-321-0987</F>
               <F>john_doe@nowhere.com</F><F>1998 Black Intrigue</F><F>BLACK</F></R>
            <R><F>Jenny Jones</F><F>69 Talkshow Road</F><F>543-210-
               9876</F><F>jjones@tv.com</F><F>1998 Red Intrigue</F><F>RED</F></R>
         </RECORDS>
      </INTERNAL_DATA>
   </DATA>
</PPMLT>
```

A.5 Example results

The table below summarizes the effect of the templating application shown in the example above. In this example, a single PPML Document requires approximately 4.5k bytes of PPML code, and the Reusable Object definitions require approximately 8k bytes. Each record of customer data is about 140 bytes.

A.5.1 PPML without templating

	Total data required to produce this many Instance Documents			
Method	**25 Instance Documents**	**1,000 Instance Documents**	**10,000 Inst. Documents**	**100,000 Inst. Documents**
PPML file size including Reusable Object definitions ("ROs")	ROs: 8k PPML excluding Inst. Docs: 10k Inst. Docs: 122k **Total: 140k**	ROs: 8k PPML excluding Inst. Docs: 10k Inst. Docs: 4.5MB **Total: 4.52MB**	ROs: 8k PPML excluding Inst. Docs: 10k Inst. Docs: 45MB **Total: 45MB**	ROs: 8k PPML excluding Inst. Docs: 10k Inst. Docs: 450MB **Total: 450MB**
PPML file size with ROs pre-loaded	PPML excluding Inst. Docs: 10k Inst. Docs: 122k **Total: 132k**	PPML excluding Inst. Docs: 10k Inst. Docs: 4.5MB **Total: 4.51MB**	PPML excluding Inst. Docs: 10k Inst. Docs: 45MB **Total: 45MB**	PPML excluding Inst. Docs: 10k Inst. Docs: 450MB **Total: 450MB**

A.5.2 With templating

	Data required to produce this many Instance Documents			
Method	**25 Instance Documents**	**1,000 Instance Documents**	**10,000 Inst. Documents**	**100,000 Inst. Documents**
PPMLT file size Including Reusable Object definitions ("ROs")	Data: 3.5k ROs: 8k Template & Mapper: 6.5k **Total: 18k**	Data: 140k ROs: 8k Template & Mapper: 6.5k **Total: 154k**	Data: 1.4MB ROs: 8k Template & Mapper: 6.5k **Total: 1.41MB**	Data: 14MB ROs: 8k Template & Mapper: 6.5k **Total: 14MB**
PPMLT file size with ROs pre-loaded	Data: 3.5k Template & Mapper: 6.5k **Total: 10k**	Data: 140k Template & Mapper: 6.5k **Total: 146k**	Data: 1.4MB Template & Mapper: 6.5k **Total: 1.41MB**	Data: 14MB Template & Mapper: 6.5k **Total: 14MB**
PPMLT file size with ROs, template, and mapper all pre-loaded	Data: 3.5k Other: 0.3k **Total: 3.8k**	Data: 140k Other: 0.3k **Total: 143k**	Data: 1.4MB Other: 0.3k **Total: 1.4MB**	Data: 14MB Other: 0.3k **Total: 14MB**

