

Aperio Markup Language (APML)

Programmer's Reference



©Copyright 2007 Aperio Technologies, Inc.
Part Number/Revision: MAN-0068, Revision B
Date: December 3, 2007

This document applies to software versions Release 8.2 and later.

All rights reserved. This document may not be copied in whole or in part or reproduced in any other media without the express written permission of Aperio Technologies, Inc. Please note that under copyright law, copying includes translation into another language.

User Resources

For the latest information on Aperio Technologies products and services, please visit the Aperio Technologies website at: <http://www.aperio.com>.

Disclaimers

This manual is not a substitute for the detailed operator training provided by Aperio Technologies, Inc., or for other advanced instruction. Aperio Technologies Field Representatives should be contacted immediately for assistance in the event of any instrument malfunction. Installation of hardware should only be performed by a certified Aperio Technologies Service Engineer.

ImageServer is intended for use with the SVS file format (the native format for digital slides created by scanning glass slides with the ScanScope scanner). Educators will use Aperio software to view and modify digital slides in Composite WebSlide (CWS) format.

Aperio products are FDA cleared for specific clinical applications, and are intended for research use for other applications.

Trademarks and Patents

ScanScope is a registered trademark and ImageServer, TMA Lab, ImageScope, and Spectrum are trademarks of Aperio Technologies, Inc. All other trade names and trademarks are the property of their respective holders.

Aperio products are protected by U.S. Patents: 6,711,283; 6,917,696; 7,035,478; and 7,116,440; and licensed under one or more of the following U.S. Patents: 6,101,265; 6,272,235; 6,522,774; 6,775,402; 6,396,941; 6,674,881; 6,226,392; 6,404,906; 6,674,884; and 6,466,690.

Contact Information

Headquarters:	Aperio Technologies, Inc. 1430 Vantage Court Vista, CA 92081 United States	European Office:	Aperio 3 The Sanctuary Eden Office Park Ham Green Bristol BS20 0DD, UK
----------------------	---	-------------------------	--

United States of America

Tel: 866-478-4111 (toll free)

Fax: 760-539-1116

Customer Service Tel: 866-478-4111 (toll free)

Technical Support Tel: 866-478-4111 (toll free)

Email: support@aperio.com

Europe

Tel: +44 (0) 1275 375123

Fax: +44(0) 1275 373501

Customer Service Tel: +44 (0) 1275 375123

Technical Support Tel: +44 (0) 1275 375123

Email: europesupport@aperio.com

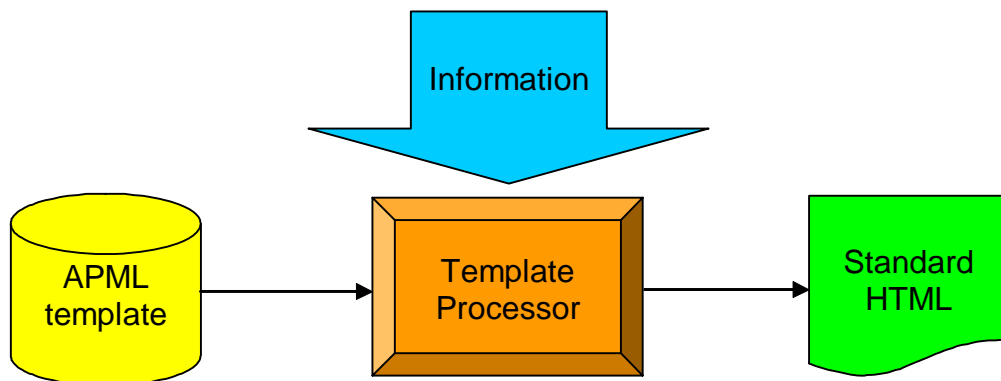
Contents

CHAPTER 1 – OVERVIEW	1
How It Works	1
Variables, Tables, and Dictionaries	3
APML Language	4
CHAPTER 2 – APML REFERENCE	5
#... comment	5
{%...\$}.....	5
&name.....	5
<%SET name=value (name2=value2 ...) %>	6
<%EVAL variable_name (parm=value (parm2=value2...)) %>	6
<%IF expression %> true_APML (<%ELSE%> false_APML) <%END%>.....	6
<%TABLE Construct.....	7
<%PRINTF format_string (operand1 (operand2 ...)) %>.....	8
<%DUMP%>	8
CHAPTER 3 – APML EXPRESSION EVALUATION	9
APML Expression Operators	10
CHAPTER 4 – APML ENVIRONMENT	11
INDEX	13

1 Overview

This document describes APML, the Aperiio Markup Language. APML is a simple superset of HTML that enables dynamic web pages to be constructed using simple tools. Aperiio's Internet applications (websites) are built using APML. In addition, Aperiio's ImageServer uses APML to enable image directories and files to be displayed using simple templates.

APML is implemented by a run-time interpreter which creates standard HTML pages for display on web browsers. Everything is done on the server; the client can be any web browser. APML pages are "templates" which are "filled in" at run-time by the interpreter to create HTML pages, as illustrated below:



The remainder of this document describes APML in detail. Everything in a template is "passed through" to the output, except for APML tags which tell the APML interpreter to insert data or perform some function. APML commands provide features like nesting (INCLUDE...) and conditional processing (IF...THEN...ELSE).

How It Works

The APML interpreter loads the requested APML template, sequentially parses it and performs variable substitution and command processing, and then generates

the output HTML. The information merged into the template comes from one or more *dictionaries*, depending on the environment. For example, when ImageServer processes an APML template it creates dictionaries with environment information, input parameters, and image directory or file information.

A sample APML template is shown below:

```

welcome.apml
#
# This is a simple APML template ❶
#
<% SET viscnt=(&cookies.visits+1) %>Set-cookie: visits=&viscnt ❶
<HTML>
<HEAD>
<TITLE>Welcome to &SERVER_NAME</TITLE> ❷
</HEAD>
<BODY>
<p>Your address is &REMOTE_ADDR [(aka "&REMOTE_HOST")] ❸
<% IF &viscnt > 1 %><p>You have visited &viscnt times! ❹
<% ELSE%><p>This is your first visit!
<% END%>
<p><% INCLUDE trailer %> ❺
</BODY>
</HTML>
    
```

When this page is requested (via an HTTP GET from a web browser), the web server will run APML passing “welcome.apml” as a parameter. The APML interpreter processes input parameters, environment variables, and cookies, then parses the APML. The highlighted sections above are APML constructs processed as the HTML is generated:

- ❶ APML templates may contain comment lines which begin with a pound sign, #.
- ❷ The `<%SET%>` command assigns a new variable value, evaluating an expression.
- ❸ The `&name` construct substitutes a dictionary value into the HTML.
- ❹ The `[...]` construct conditionally generates HTML if a value is defined.
- ❺ The `<%IF%>...<%ELSE%>...<%END%>` commands cause conditional processing.
- ❻ The `<%INCLUDE%>` command processes another template recursively.

This example illustrates some of the simplicity and power of APML. Read on to learn more!

Variables, Tables, and Dictionaries

APML manages information in *dictionaries*. The simplest kinds of information are *variables*, which have a name and a value. Dictionaries can also contain *tables*, which are two-dimensional arrays of information, and even other dictionaries.

When an APML page is processed, several dictionaries are created and filled with information from the environment (see Chapter 4, “APML Environment,” on page 11). Subsequently, APML commands can add, delete, or change information.

Here are some examples of variables that might be stored in a dictionary:

Name	Value
user_number	2150
name	Ole Eichhorn
address	4310 Golf Course Drive
city	Westlake Village
state	CA
zip	91362
bank_balance	
image	/My Images/kidney 101 40X.svs

Each variable has a name and a value. The name can be any string with letters, numbers, or underscores. Names are not case sensitive (“name” is the same as “NAME” or “Name”). The value can be any string whatsoever, of any length, with any characters, including leading, embedded, and trailing spaces. Values can be zero-length (“null”), as with **bank_balance** above.

Certain kinds of information lend themselves to a two-dimensional tabular organization. For example:

table name = image_attributes

Image	Width	Height	Date	Compressed_Size
kidney 101.svs	40200	30700	05/01/02	300324
jim00123.svs	60340	42560	04/01/02	456200

table name = scanscopes

ID	Name	Made_Date	Made_By	Notes
1	Elite machine	03/20/02	Russ	
2	Vista machine	04/15/02	Brian	makes funny noises
5	MGH machine	05/01/02	Fili	coolest so far

Each table of information has a name, and one or more named *columns*. The name of a table and the names of the columns can be anything (following the same conventions as variable names). Each table has zero or more rows of information. Each row could represent an image file or a ScanScope (as above), or anything else. The values in each column of each row are variable values — they can be any string of characters, including nothing (as with the “notes” column of the first row of the “scanscopes” table above).

Dictionaries may also contain other dictionaries. When a lookup into a dictionary is performed, the dictionary is searched for a matching value. If the value is found, it is returned, otherwise each of the dictionaries stored in the main dictionary are searched in turn, and so on recursively. A specific dictionary can be indicated with a “.” in a variable name, as **&dictionary.variable**. For example, for the substitution **&parms.limit**, the dictionary named “parms” is searched for a variable named “limit.”

APML Language

As with HTML, APML is not case sensitive. APML “commands” are often capitalized (as in this document), but this is to enhance legibility and is not required. Names of variables, dictionaries, etc. are likewise not case sensitive.

Within APML constructs, it is sometimes necessary to group text. Either single or double quotes may be used; in both cases, the delimiting quote characters are discarded and do not become part of the text. To embed single or double quotes inside a text string, precede the quote character by a backslash. When text is grouped by double quotes, any APML constructs within the text are evaluated normally. When text is grouped by single quotes, the grouped text is used “as is,” and no further APML variable substitution or command processing is performed.

Certain characters have meaning to APML, such as “&” (which introduces a variable substitution). Any character can be preceded by a backslash to make it “transparent” to APML; backslashes are dropped in the output HTML. When in doubt, use a backslash.

The following chapter discusses each APML construct in more detail.

2

APML Reference

This chapter lists all of the APML language constructs.

#... comment

APML comment lines can be inserted anywhere in a template file. A # character at the start of a line indicates a comment; the entire line is dropped and nothing appears in the output HTML.

{%...\$}

This construct allows *anything* to be passed straight through to the output. Any APML constructs located between the braces are ignored, including variable substitution, commands, quotes, etc. The braces are dropped when output is generated.

To include a {% construct in the output, precede the brace with a backslash.

&name

This construct can occur anywhere in the template file, and causes substitution by the corresponding value. If **name** is not defined in the dictionary, a null value is substituted. If desired, a specific dictionary within the main dictionary can be searched with the syntax:

```
&dictionary.name
```

This process can be repeated to look up within deeply nested dictionaries, as in:

```
&web.cookies.mycookie
```

This looks for a value named **mycookie** located in the dictionary named **cookies** which is located in the dictionary named **web**, which is located in the main dictionary.

Spaces or other punctuation usually terminates data names, but a semicolon can be used in situations where it is necessary to indicate explicitly where the data name ends. For example, if the value of a variable named **meat** is **ham**, you could make **hamburger** as follows:

```
&meat;burger
```

<%SET name=value (name2=value2 ...) %>

This construct defines one or more variables. This allows an APMML template to "override" a value previously supplied by a program, and/or to set a default value prior to processing another template or calling a function. Values set within a template file are passed to subsidiary templates processed via the <% INCLUDE ... %> construct.

If the "value" is enclosed by parentheses, it is assumed to be an expression and is evaluated, and the result is assigned to the variable. For example, assume the variable **page** has the value 4, and the variable **temp** has the value 73. Then:

```
<%SET myage=&page+&temp %> ... causes "myage" to have value "4+73"
<%SET myage=( &page+&temp ) %> ... causes "myage" to have value 77
```

When parentheses are used in this way, any spaces or other delimiters may appear in the expression without ending the value. The second example above could be given as:

```
<%SET myage=( &page + &temp ) %>
```

Note that all substitution is performed first, and then the SET is processed. For example:

```
<%SET x=1 %><%SET x=3 y=( &x+5 ) %>
```

In this example the value of **y** will be 6, because the **x=3** is processed *after* substitution.

<%EVAL variable_name (parm=value (parm2=value2...)) %>

This construct causes the *value* of the specified variable to be interpreted as APMML. Essentially a string variable can be a function, and can be "called" with <%EVAL%>. The [optional] parameter values specified are set before the APMML in the variable value is processed, and then un-set. Here's an example:

```
<%SET myfunc='test val = &test' %>
<%EVAL myfunc test=1%>
<%EVAL myfunc test=2%>
```

The first line defines the variable **myfunc**, assigning a value which is some APMML. The single quotes enclosing the value prevent it from being interpreted. The second line causes the value of **myfunc** to be evaluated. The output will be **test val = 1**. The third line's output will be **test val =2**.

<%IF expression %> true_APMML (<%ELSE%> false_APMML) <%END%>

This construct allows for conditional inclusion of APMML. The "expression" is evaluated, and if the value is true then **true_APMML** is processed, and the resulting value substituted into the generated HTML. If the expression value is false and the optional <%ELSE%> is specified, then **false_APMML** is processed instead. Normal processing resumes after the <%END%>. Both **true_APMML** and

false_APML can contain any valid APML, including other `<%IF%>`s; IF constructs can be nested to any reasonable depth. The details of APML expression evaluation are given in the next chapter.

An alternate form of the IF construct which is syntactically more compact is the use of square brackets, as follows:

```
[ APML_string ]
```

This construct may appear anywhere in an APML template. The **APML_string** is substituted normally, except that if an unresolved reference to a name/value appears anywhere in **APML_string**, the entire string between square brackets is dropped. For example:

```
[ <b>The apparent magnification is &appmag</b><p> ]
```

If **appmag** is defined, the value is substituted and the string between the square brackets becomes part of the output. If **appmag** is not defined, the entire string is dropped. This notation is especially useful for conditional generation of table columns, see below.

<%TABLE Construct

The table construct takes the form:

```
<%TABLE table_name
    (HEADER='APML_for_header' )
    DATA='APML_for_row'
    (DATA2='APML_for_2nd_row' ) ... %>
```

This construct causes the specified data table to be formatted. If specified, the HEADER parameter is used to format the header row of the table. The 'APML_for_header' parameter may contain any valid APML (typically with embedded `<TH>` tags), and may include value substitution references. Columns of the table can be referenced by name as well. The DATA parameter is used to format the data row(s) of the table (typically with embedded `<TD>` tags). If more than one DATA parameter is supplied (DATA, DATA2, DATA3, etc.), they are used in a round-robin fashion for successive table rows (this is to support alternating colors and/or formats). HEADER and DATA parameters are usually specified enclosed by single quotes, because the parameter values should be evaluated when the table is processed, not when the `<% TABLE ... %>` construct is parsed.

The **APML...** parameters can contain square brackets to indicate an optional construct (see IF discussion above). For example:

```
DATA=' <TD>&name</TD>[ <TD>&title</TD> ]<TD>&value</TD> '
```

In this example, the `<TD>&title</TD>` would only be processed if "title" actually existed as a column in the table.

As with ordinary values, tables can be looked up explicitly in a particular dictionary by using the "dictionary.name" syntax.

```
<%INCLUDE filename (name1=value1... (name2=value2)... ) %>
```

This causes the specified APML template file to be processed, and the generated output HTML is substituted for the construct. Any **name=value** parameters specified are added to the dictionary before processing the file, and then are removed. Template files can be nested in this fashion to any reasonable depth.

This construct allows common syntactic elements like headers, footers, error messages, etc. to be formatted by common externally defined APML.

<%INCLUDE%>ed files are often given the file extension .apml to indicate their contents.

<%PRINTF format_string (operand1 (operand2 ...)) %>

The PRINTF construct enables printf-style formatting of operands. The *format_string* may include any valid **printf** formatting characters. The operands may be strings or numeric values which are substituted into the formatted string. The entire construct is replaced by the formatted output. If an error occurs during formatting, an error message replaces the construct.

<%DUMP%>

This construct is useful for debugging APML templates; it formats all presently defined variables, tables, dictionaries, functions, etc. into the output HTML. This can be done conditionally, for example:

```
<%IF "&mycookie" != "expected" %><XMP><%DUMP%></XMP><%END%>
```

Examining the output from this command is a good way to become familiar with the tables and dictionaries created as part of the APML environment. It is a good idea to bracket the <%DUMP%> with <XMP> and </XMP> (as shown above), which cause the formatting of the DUMP output to be preserved by the HTML.

A simple “what’s going on here” APML page can be made as follows:

<pre>test.apml <HTML> <BODY> <XMP><%DUMP%></XMP> </BODY> </HTML></pre>
--

This can be useful for examining directory and file information, input parameters, etc.

3

APML Expression Evaluation

APML expressions can appear anywhere *within* APML tags. Any text string that is enclosed by parenthesis is automatically evaluated as an expression.

Before processing an expression, the entire expression will be substituted. That is, any APML references of the form `&name`, `<% function %>` and/or conditional references [...] will be processed and resolved. The result will be a string containing three types of syntactic elements:

Numeric operands	any sequence of digits. Leading signs are allowed, as are embedded commas or decimal points (all arithmetic is 64-bit floating point).
String operands	any sequence of characters delimited by single or double quotes. Single quotes prevent APML substitution from occurring, double quotes do not.
Operators	specific operators, as described below.

The table below lists all supported operators, listed in the order of precedence. Parentheses may be used to override the order of precedence.

When arithmetic values are interpreted logically, any nonzero value is **true** and zero is **false**. When string operands are used with arithmetic or logical operators, their value is taken as the string's length in characters. Null strings have a zero length and hence an arithmetic value of 0 and a logical value of **false**. When string operands are used with relational operators, the string lengths are compared. If both strings have the same length, their contents are compared using the ASCII collating sequence.

APML Expression Operators

Category	Operator	Description
Arithmetic Operators	*, /, %	Multiplication, division, and modulo
	+, -	Addition and subtraction
Relational Operators	=, EQ, !=, NE, <, LT, <=, LE, >, GT, >=, GE	Equal, not equal, less than, less than or equal, greater than, greater than or equal
Logical Operators	!, NOT	Logical NOT
	&, AND	Logical AND
	, OR	Logical OR

Whitespace (spaces, tabs, carriage returns, and line feeds) may be placed anywhere within an expression and does not affect expression evaluation.

When expressions are evaluated, they always have a numeric result (which can be interpreted as a logical result using zero=FALSE, non-zero=TRUE). If the result is referenced as a string, it will be formatted as a decimal integer.

4 APML Environment

This section describes the information that is gathered from the environment and stored in variables, tables, and dictionaries¹.

The following variables are created automatically:

Variable	Description
<code>_nest</code>	Current template nesting depth
<code>_path</code>	APML template directory path
<code>_parmc</code>	Number of input parameters

The following tables are created automatically:

Table	Columns	Description
<code>_strings</code>	name, value	one row for each variable
<code>_tables</code>	name, columns, rows	one row for each table
<code>_funcs</code>	name	one row for each function
<code>_dicts</code>	name, strings, tables, funcs	one row for each dictionary
<code>_apml</code>	type, source	one row for each nested source
<code>_parmv</code>	name, value	one row for each input parameter

¹ Additional information may be stored in dictionaries depending upon the environment. For example, `ImageServer` stores request information and directory or image file information (see the *ImageServer Programmer's Reference* for details).

Index

- APML
 - case sensitivity, 4
 - interpreter, 1
 - reference
 - #, 5
 - %DUMP, 8
 - %EVAL, 6
 - %IF, 6
 - %name, 5
 - %PRINTF, 8
 - %SET name, 6
 - %TABLE, 7
 - {%...\$}, 5
 - comments, 5
 - conditionals, 6
 - debugging, 8
 - defining variables, 6
 - evaluating expressions, 6
 - formatting, 8
 - formatting data tables, 7
 - interpreting value of variable
 - as APML, 6
 - substitutions, 5
 - special character handling, 4
 - dictionaries, 3
 - environment, 11
 - expression evaluation, 9
 - interpreter, 1
 - operators, 10
 - overview, 1
 - sample template, 2
 - tables, 3
 - automatically created
 - _apml, 11
 - _dicts, 11
 - _funcs, 11
 - _parmv, 11
 - _string, 11
 - _tables, 11
 - variables, 3
 - automatically created, 11
 - _nest, 11
 - _parmc, 11
 - _path, 11

Aperio Markup Language (APML) Programmer's Reference

MAN-0068, Revision B