Title: igpp.docgen tool Author: Todd King Date: May 10, 2012 CSS: style.css

igpp.docgen tool

Last update: May 10, 2012; *Author: Todd King*

Introduction

The igpp.docgen executable jar is a set of tools to parse structured information (metadata) and merge the information with an Apache Velocity template.

A Velocity template contains text and references to variables. The igpp.docgen executable is used to defined the value assigned to variables. Variables can be defined on the command line, read from a PDS3 label or from a text file. Variables are placed into one or more named contexts. When loading variables from a file the context name can be chosen. Variables from the command line are placed in a context named "options". The syntax for specifying a named context that is populated from a file is:

```
name:file
```

files ending in ".lbl" are parsed as PDS3 label files. All others are parsed as text files containing one keyword=value per line. Lines beginning with "#" are considered comments.

Usage: java igpp.docgen.Process [options] [file...]

Options:

```
-3,--pds3 <arg>        PDS3. Parse file as though it is in PDS3 format.
-a,--table <arg>       Table. Parse file as though it is in table format.
With the table format values are
expressed as rows with the first line being the varaible name.
-f,--format <arg>      Format. Format output with a given style. Allowed values are
PDS3 and XML. Default: XML
-h,--help              Dispay this text
-i,--include <arg>     Include Path. Path to look for files referenced with an INCLUDE
or STRUCTURE pointer.
-l,--list <arg>        List. Parse file as though it is in list format. With the list
format values are expressed as keyword=value, one per line.
-o,--output <arg>      Output. Output generated document to {file}. Default: System.out
-s,--separator <arg>   Separator. Pattern that separates values in tabular files.
Default: is a tab ()
-t,--template <arg>    Template. The template folder to search for templates file.
-v,--verbose           Verbose. Show status at each step.
```

Variable Names in "options"

The name for each variable in the "options" context matches the long name of the option. Additional variables can be defined using the syntax:

```
name=value
```

Installation

---

igpp.docgen is distributed as a executable JAR file. You can download the current distribution from To "install" an executable JAR copy the JAR file to a local directory. It can be run using the "-jar" option to the "java" command. For example:

```
java -jar igpp.docgen.jar [options]
```

The igpp.docgen.jar file can also be installed in a location that is in the Java extension directory. When installed this way the tool can be run with the "igpp.docgen.Process" class. For example:

```
java igpp.docgen.Process [options]
```

Usage Examples

Here are a few examples using the exectable jar:

Use the Velocity Template "example-1.vm" and the file "table-1.csv" assigned to the context "table"

```
java -jar igpp.docgen.jar table:table-1.csv example-1.vm
```

will replace parameters in "example-1.vm" with values from "table-1.csv" and output the result as formatted XML.

Do the same with lot's a diagnostic information

```
java -jar igpp.docgen.jar -v table:table-1.csv example-1.vm
```

Do the same, but look or templates in a directory called "/templates".

```
java -jar igpp.docgen.jar -v -t /templates table:table-1.csv example-1.vm
```

Used the Velocity Template "example-pds3.vm" and the file "table-1.csv" assigned to the context "table"

```
java -jar igpp.docgen.jar -f pds3 table:table-1.csv example-pds3.vm
```

will replace parameters in "example-pds3.vm" with values from "table-1.csv" and output the result as formatted as a PDS3 label.

Common Velocity

The Velocity Template Language (VTL) is embedded mark-up which can be included in text files and interpreted by the Velocity processor. It allows the content of documents (like XML, web pages, etc) to be parameterized. Some commonly used elements of VTL are:

Variables

A variable consists of leading "$" followed a reference name of the variable. A reference name is a sequence of alphanumeric characters. A reference name may also contain a dash or underscore character, but must begin with a alphanumeric character.

Examples:

```
$example
$example-another
```

igpp.docgen parses a variety of file formats (i.e., text, PDS3 label) and generates variable definitions to reflect the content. Variables are placed in named context which can be referenced using a dot notation. For example, if the variable "this" is placed in the "my" context it can be referenced with the syntax:

```
$my.example
```

If a variable does not exist the reference string is output by Velocity. To have a blank string appear you can use a "silent" reference. A silent reference is one with a exclamation point (!) following the dollar sign ($). For example:

```
$!my.example
```

Will output the value of "my.example" if it exists and blank string if it does not.

In the preceding examples the shorthand notation for variables was used. If the value of variable is to be embedded in another string you can use the format reference notation which quotes the variable name in curly braces. For example:

```
${my.example}
```

Methods

A reference may also refer to a method (function) associated with the class of variable. In docgen values are either a String, ArrayList of String or a HashMap.

Some commonly used methods are:

```
HashMap.size() : The number of items in the HashMap.
ArrayList.size() : The number of items in the ArrayList.
String.length() : The length of the String.
```

Data Type Conversion

It is sometimes necessary to convert the data type of a value from a String to an Integer or double in order to perform math on the value. Two contexts are defined for this purpose.

```
Integer: Perform actions on integers.
Uses the java.lang.Integer class.
$Integer.parseInt($value) will transform a the string assigned
to $value into an int.
Double: Perform actions on doubles.
Uses the java.lang.Double class.
$Double.parseDouble($value) will transform a the string assigned
to $value into an double.
```

File Information When generating metadata it is often necessary to determine some information about the file such the size of last modificaiton data. A context called "File" which is tied to the igpp.util.File class is defined. With the "File" context the size of a file can be obtained with a call like:

```
$File.getSize(pathname)
```

An MD5 checksum for the file can be obtained with a call like:

```
$File.getMD5(pathname)
```

See the igpp.util.File documentation for more details.

Date Conversion Time values used in metadata often must confirm to a particular format. The most commonly used format is called ISO8601. A context called "Date" is tied to the igpp.util.Date class. With the "Date" context you can convert time values in just about any format to the ISO8601 format with the call:

```
$Date.getISO8601DateString(string)
```

You can also get the time string for the current moment with a call to

```
$Date.getNow()
```

See the igpp.util.Date documentation for more details.

Math

Mathematical expressions are evaluated when defining a variable using #set and in #if statements. To use a calaculated value, for example based on the sum or two other values, you must define a new varaible using #set and then use the new variable in the desired context. For example, the sum of the value $offset and $length.

```
#set($total = Integer.parseInt($offset) + Integer.parseInt($length))
```

Flow Control

The Velocity Template Language supports flow control while processing the content of a template. There are two commonly used methods. One is the "if" control and the other is the "foreach" control. In the Velocity Template Language directives, like "if" and "foreach" have a "#" prefix.

if

With the "if" control actions or sections of a document can be included based on a logical comparison. For example, if a variable has a certain value.

Example:

```
#if($my.example == "yes")
Include this in the output.
#end
```

You can also use "if" to determine if a value is an array or a single value by checking for the existence of the "size()" method.

Example:

```
#if($my.example.size())
It's an array.
#end
```

foreach

With the "foreach" control a set of actions can be taken for every item in an array. If the variable $my.example were an array then each item can be assigned to a local variable and the local variable can be used within a block a text.

Example:

```
#foreach ($item in $my.example)
Do this for $item.
#end
```

Methods

A variable may also have methods (functions) than can perform some task. Some commonly used methods for arrays are:

Determine number of elements in an array called "$my.example":

```
$my.example.size()
```

Additional information about Velocity

Common Context

There are three common contexts created that can be used in templates. For complete details of all available methods check the documentaiton for each class. The most useful and commonly used methods are:

Integer : Provides access to the java.lang.Integer class.

```
int Integer.parseInt(string) : Parses the string as an integer.
Throws an exception if the string is not a valid integer value.
String toBinaryString(int i) : Returns a string representation of the
integer argument as an unsigned integer in base 2.
String toHexString(int i) : Returns a string representation of the
integer argument as an unsigned integer in base 16.
String toOctalString(int i) : Returns a string representation of the
integer argument as an unsigned integer in base 8.
```

Double : Provides access to the java.lang.Double class.

```
double Double.parseDouble(string) : Parses the string as an double.
Throws an exception if the string is not a valid double value.
```

String : Provides access to the java.lang.String class.

```
startsWith(prefix) : Returns true if string starts with "prefix", false otherwise.
toUpperCase() : Converts all of the characters in this String to upper case using the ru
toLowerCase() : Converts all of the characters in this String to lower case using the ru
trim() : Returns a copy of the string, with leading and trailing whitespace omitted.
replaceAll(String regex, String replacement) : Replaces each substring of this string th
```

File : Provides access to the igpp.util.File class.

```
copy(String source, String destination) : Copy a file from one
location to another.
rename(String source, String destination) : Rename a file.
String getName(String pathname) : Extract the name of the file from
a pathname.
String getParent(String pathname) : Extract the parent of the file from
a pathname.
String getBaseName(String pathname) : Extract the base name of the file
from a pathname.
String getExtension(String pathname) : Extract the extension of the file
from a pathname.
long getSize(String pathname) : Return the size of the file in bytes.
String getMD5(String pathname) : Return the MD5 digest for a file.
String getSHA1(String pathname) : Return the SHA-1 digest for a file.
String getSHA256(String pathname) : Return the SHA-256 digest for a file.
String getSHA512(String pathname) : Return the SHA-512 digest for a file.
```

Date : Provides access to the igpp.util.Date class.

```
now() : return the current data and time in the locale format.
now("ISO") : return the current date and time in ISO-9601 format.
```

Calc : Provide access to the igpp.util.Calc class.

```
perform(arg1, op, arg2) : Perform the mathematic operation (-, +, *, /) on arg1 and arg2
sum(list) : Sum the values in the list.
ceil(value) : Round a value up to nearest integer and return value as a long.
floor(value) : Round a value down to nearest integer and return value as a long.
```

Text file parsing (.txt)

Files with the extension ".txt" are considered a text file. When text files are parsed lines beginning with "#" are considered comments. All other lines are parsed as a "name=value". If a line does not conform to the "name=value" syntax it is ignored. If a value is enclosed in curly braces {} the value is parsed as a comma separated list.

Lookup Tables Keyword/Value text files can be used to create simple lookup tables. The value associated with a keyword can be retrieved with a

```
$context.get(name)
```

where "context" is the defined context in Velocity and "name" is the keyword to find. "name" can also be a reference to parameter in another context. For example, to retrieve the value from a "target" context using the value for a spacecraft in a "list" context do the following:

$target.get($list.spacecraft)

Delimited table file parsing (.csv, .tab)

Files with the extension ".csv" or ".tab" are parsed as a delimited table. Field names are taken from the first record. Comments are lines that begin with "#" and are ignore. The array of records is assigned the name "record" in the defined Velocity context. The number of records can be determined with record.size().

PDS3 Label Parsing (.lbl)

Files with the extension ".lbl" are considered a PDS3 label file. When a PDS3 label is parsed a variable hierarchy is created which matches the organization of objects in the label. If multiple objects occur at the same level an array is created in the variable tree.

The PDS3 label parser is a Object Definition Language (ODL) parser and is very lenient as to compliance with the PDS3 data model schema. You can use ODL to define nested values using the "OBJECT=name" to nest values.

All OBJECT definitions create a new array in the generated Velocity context. This is true even if there is only one occurrence of an OBJECT definition. In the Velocity template use a "#foreach" directive to process each OBJECT.

Data File

A detached label has the same base name as the data file it describes. Both the label and the data file must be located in the same folder. When the PDS3 label is parsed the file system is search for the corresponding data file. Attributes for the located file are included with in the top level variable for the label. The added attributes are:

```
PRODUCT_FILE: The file name of the data file.
PRODUCT_FILE_MD5: The MD5 checksum for the data file.
```

Pointers

PDS3 supports pointers which are named references to a file which are tied to an OBJECT. A pointer has the general form:

^{name}_{class}=({file}, {location})

where {name} is a unique prefix for the object refernce, {class} is the object class (i.e., FILE), {file} is the name of the file and {location} is the location within the file to the start of the object.

When a PDS3 label is parsed the class of the pointer object is used for the variable reference with attributes added to the variable to describe the pointer and the referenced file. The attributes included in the variable are:

```
OBJECT_NAME: The name portion of the pointer.
OBJECT_FILE: The file referenced by the pointer.
OBJECT_FILE_MD5: The MD5 checksum for the file.
OBJECT_LOCATION: The location to the start of the object from the start of the file.
```

Example:

```
/*Band A */
^HFRA_CAL_LEVELS_AGC_TABLE   = ("T2007002_CAL0.TAB", 4) /* Agc */
OBJECT                       = HFRA_CAL_LEVELS_AGC_TABLE
INTERCHANGE_FORMAT        = ASCII
ROW_BYTES                 = 140
ROWS                      = 4
COLUMNS                   = 2
DESCRIPTION               = ""
OBJECT                    = COLUMN
NAME                   = CALIBRATION_LEVEL
DATA_TYPE              = CHARACTER
START_BYTE             = 1
BYTES                  = 6
DESCRIPTION            = "HFR internal calibration active antenna stimulus."
END_OBJECT                = COLUMN
END_OBJECT                   = HFRA_CAL_LEVELS_AGC_TABLE
```

The file associated with the TABLE in this example will be referenced as:

```
$label.TABLE.Product_file
```

References

Introduction to Velocity

NOAA Velocity User Guide

Apache Velocity Developer Guide