Installing and Configuring Validate Tool

Validate Tool Functions

The primary functions provided by *Validate Tool* as of this writing are schematic validation (against both XSD and Schematron files), and referential integrity checking of collections and bundles (i.e., ensuring all member products are present and complete, and no non-member products are present). Capabilities still in development include the ability to validate data objects (like images and tables) against their label definitions.

Validate Tool can also, on request, perform a checksum verification as part of the validation process.

Goal

Our goal for this page is to start with the Validate Tool installation package and end up with the tool installed for general use on the target system. "General use" in this case means you can invoke the tool in any directory where you happen to be working with a command line that, in its simplest form, looks something like this:

```
% validate product.xml -r report.txt
```

Note that there is an installation document with a more terse installation process description included in the documentation provided with the code. Those experienced with installing Java-based software on their system may well prefer to reference that. This page is intended for those who are unfamiliar with that process and would like a bit of additional detail and guidance.

Part List

To run the Validate Tool locally, you'll need:

- **The Validate Tool package**. The package is available as either a ZIP file or a compressed TAR file from the PDS Validate GitHub page, https://nasa-pds.github.io/validate/. Either format will do.
- **Java 6 (1.6) or later**. Type `java -version` at your command line to what version of Java, if any, you have available. If you don't have Java installed, or want to work with a later version, you'll usually need administrator privileges on your computer to download and install a newer version from the Oracle web site https://java.com/download. Java 7 (1.7) and later includes a handy feature that will help with configuration later on, so if you are still running a (relatively) ancient version, you now have one more reason to upgrade.
- **A text editor** that can handle simple text files for batch processing without filling them up with stupid control characters. On linux-based systems, things like `vi`, `pico`, or `gedit` will work; from the Windows DOS command line, you can use the `edit` command on older systems (pre-Windows 7), or *Notepad* (which can be invoked from the Windows command line as `notepad`) on newer ones.

General Procedure

Here's the general procedure for setting up the tool:

1. Unpack the Validate Tool package.
2. Move the directories you need to run the tool to a permanent location.

3. Edit the wrapper script for the local environment.
4. Install the wrapper script.
5. Test the installation.
6. Rejoice in the knowledge of a job well done.

Procedure

Unpack the Validate Tool Package

Use a standard ZIP tool (*unzip* on linux-based systems; the *Extract All* option in *Windows Explorer*) to extract the files from the ZIP package. For the tar file, use the *z* option to uncompress while you extract on a linux system. You should end up with a directory having a name that starts with *validate-* and ends with the internal version number of the tool. As of this writing, the latest version of the tool is 1.11.0, so the delivery package unpacks into a directory called *validate-1.11.0*. You can unpack it anywhere - we'll move the directory tree we need to a new home once we're picked one out. If you haven't inspected previous *Validate Tool* delivery packages, you should probably take a few minutes to familiarize yourself with the contents.

Here's what you'll find in the unpacked directory:

Executables

The executable elements of the package include:

bin/
You'll only need one of the two files from this directory, depending on your system type. We'll be modifying the appropriate script to work on your local system.

lib/
This directory contains the Java archive files comprising the *Validate Tool* code.

Documentation

The **doc/** subdirectory contains an HTML directory tree. Point your browser to the *index.html* file to see it in its intended format.

Peanuts

Like packing peanuts, these files are included in the ZIP but are not directly involved in program operation:

- *LICENSE.txt* : Standard boilerplate license (JPL employees produced this code, and JPL is part of the California Institute of Technology)
- *README.txt* : This file just directs you to the *doc/index.html* file.

Install the Executable and Support Directories

Unless you are seriously hardcore about running Java applications, you will be running Validate by invoking a wrapper script (or batch file). This script sets up some environment variables and then invokes Java with the appropriate options and arguments for executing the *validate* ".jar" file with options and arguments passed on by the wrapper script.

Note: The Java code not only expects to find the environment variables set by the wrapper script, it also expects to be running from a *bin/* directory that is adjacent to the *lib/* directory containing the *jar* files - so that aspect of the directory tree must be preserved for the code to

execute successfully. There are various ways of accomplishing that in different operating environments if you know what you are doing. This page describes one simple way to achieve that for non-experts.

Choosing an Installation Location

On linux-based multi-user systems, you can install Validate for general use by all users either by installing into one of the standard system locations (*/usr/share*, for example), or in shared disk space. If the latter, users wanting to execute Validate will likely have to add the appropriate directory location to the $PATH setting. Alternately, you as a single, non-super user can install it into your own *~/bin/* directory. Note that if you haven't created or used a personal *~/bin/* directory before, you may have to add it to your $PATH to use it.

In any event, on a linux-based systen you will ultimately have to choose one of these options:

1. Add the *validate-[version]/bin* directory to your $PATH, which requires editing your shell resource file; or
2. Create a link to *validate-[version]/bin/validate* (i.e., to the script rather than just the directory) in a directory already in your $PATH, which requires an additional edit to the *validate* wrapper script; or
3. Type the full path to the *validate* script every time you want to run it.

On Windows systems, you can install the Validate directory tree into the "Program Files\" directory for general use (this typically requires admin privileges), or in your own directory space for personal use. In either case you will have to modify the %PATH% environment variable setting information to make the *validate.bat* executable visible to all users or to yourself, respectively. Or you can run the batch file by typing the full path reference each time.

What to Copy/Move

Create a directory in your chosen installation location to hold the Validate Tool tree. You can name this *validate*, or include a version number, or rename it anything convenient. The name of the directory itself is not significant to the code.

Under this directory, copy over the entire contents of the *lib/* directory, and either copy or create a *bin/* directory to contain the edited wrapper script - *validate* for linux-based systems, or *validate.bat* for Windows systems. For linux systems, you should also make sure the *validate* script is executable.

At this point you may also want to copy over the contents of the *doc/* directory, for easy reference; it is not needed to run the code. I also copy the *README* and *LICENSE* files from the root of the install package, just in case I want to find them again later.

Edit the Wrapper Script/Batch File

The *validate* script (linux) or *validate.bat* file (Windows) is used to run the tool. This file will need to be edited to conform to the installation environment. Any simple text editor (as described above) can do the job.

Windows Batch File *validate.bat*

You'll likely want or need to make a couple changes to this file. Lines beginning with "**::**" are comments - feel free to add more.

The first executable line in the file is:

```
@echo off
```

which stops the shell from printing every executable line to your command window when you run the batch file. Comment this line out if you are trying to trouble-shoot the batch file.

Immediately after this `@echo off` line, you should probably add this line:

```
SETLOCAL
```

This makes sure that any variables that are set by this batch file do not permanently overwrite any environment variables with the same name that might have already existed for other reasons.

Following the next set of comments you will see the (uncommented) lines that check whether the *%JAVA_HOME%* environment variable is already set, and kill the batch file if it isn't. See the Finding and Setting JAVA_HOME page for detailed steps to check the variable, and to find the right value to use if it isn't set.

If *%JAVA_HOME%* is not currently set, you have two options:

1. Permanently add the definition to your environment variables. (See, for example, How to set the path and environment variables in Windows, by ComputerHope.com). In this case you don't need to make any changes to the *%JAVA_HOME%* test in the batch file.
2. Replace these lines in the batch file:

```
if not defined JAVA_HOME (
echo The JAVA_HOME environment variable is not set.
goto END
)
```

   with something like this line:

```
set JAVA_HOME="C:\Program Files\Java"
```

   where you replace *C:\Program Files\Java* with the actual location of your Java home directory. Note that there should be no spaces around the "="; and the double quotes are not necessary in the `set JAVA_HOME` line if your path does not contain embedded blanks.

Finally, the last executable line in the batch file before the `:END` statement looks like this:

```
"%JAVA_HOME%"\bin\java -Xms256m -Xmx1024m -jar "%VALIDATE_JAR%" %*
```

Remove the quotes from around %JAVA_HOME%. If the quotes were needed to set the value, then they are already part of the string and the additional quotes will cause a syntax error.

> **N.B.:** Paths with embedded blanks that are missing quotes, and extra sets of quotes, can both cause failures, frequently with messages about unexpected information or invalid paths. If you see that sort of message when you test the batch file, comment out the `@echo off` line so you can see exactly where the script is failing, and you may have to add or remove quotes on that line or an earlier line to adjust for the actual paths in your environment.

Linux *validate* script

If your *$JAVA_HOME* environment variable is not already set, the script will exit without invoking *Validate*. See the Finding and Setting JAVA_HOME page for gory details on determining the right value to set and how to set it in your environment. Note that the *validate* script is written to be run in the Bourne shell, so use Bourne shell syntax to set *$JAVA_HOME* in the script regardless of what your login shell is. So if you prefer to set *%JAVA_HOME* in the script, replace these lines:

```
if [ -z "${JAVA_HOME}" ]; then
   echo "The JAVA_HOME environment variable is not set." 1>&2
   exit 1
fi
```

with something like this line:

```
JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.131-2.b11.el7_3.x86_64/jre
```

where /usr/lib/jvm/java-1.8.0-openjdk-1.8.0.131-2.b11.el7_3.x86_64/jre should be replaced with your actual Java home directory. Alternately, if you have a handy little *java_home* script installed (see Finding and Setting JAVA_HOME), this will also work:

```
JAVA_HOME=`java_home`
```

If you are planning to link the *validate* script into a *bin/* directory (as opposed to adding a new element to your *$PATH* to access this one executable), you'll need to edit a couple more lines in the *validate* wrapper script. The script crawls the local directory tree to find related *lib/* directory using system functions, but that doesn't quite work of the script was invoked via a link. So in this case, replace these lines:

```
SCRIPT_DIR=`dirname $0`
PARENT_DIR=`cd ${SCRIPT_DIR}/.. && pwd`
```

with something like these lines:

```
PARENT_DIR=/usr/share/pds4tools/validate/validate-1.11.0
SCRIPT_DIR=${PARENT_DIR}/bin
```

where you should replace /usr/share/pds4tools/validate/validate-1.11.0 with the absolute path to your installed *validate* tree (the directory containing the *bin/* and *lib/* subdirectories). Also note the inverted order of the definitions.

Finally, make sure the *validate* script is executable.

Testing

To make sure the batch file or script can properly invoke the tool, you can run it from its *bin/* directory home. At the command prompt, go to the directory holding the *validate* script or batch file, and do:

```
validate --version
```

The response should something look like this:

```
Validate Tool
Version 1.11.0
Release Date: 2017-04-03 11:57:19
Core Schema: PDS4_PDS_1800.xsd
Core Schematron: PDS4_PDS_1800.sch

Copyright 2010-2017, by the California Institute of Technology.
ALL RIGHTS RESERVED. United States Government Sponsorship acknowledged.
Any commercial use must be negotiated with the Office of Technology Transfer
at the California Institute of Technology.

This software is subject to U. S. export control laws and regulations
(22 C.F.R. 120-130 and 15 C.F.R. 730-774). To the extent that the software
is subject to U.S. export control laws and regulations, the recipient has
```

```
    the responsibility to obtain export licenses or other export authority as
    may be required before exporting such information to foreign countries or
    providing access to foreign nationals.
```

Once you have had some experience with running *validate* and have developed preferences for options or schema file references, you may want to further modify the script or batch file to customize tool behavior accordingly.

Install the Wrapper Script/Batch File

Assuming, of course, that you don't want to do all your validation work in the *validate* tool *bin/* directory, the last step is making sure you can invoke *validate* from wherever you will be working. For Windows users this will almost certainly mean adding a new directory to you *%PATH%* environment variable. Linux users can do likewise in their *$PATH* environment variable, but also have the option of adding a link to the *validate* script in a directory already in the command path.

You can, of course, execute the script/batch file by using its full (absolute or relative) path on the command line. For ease of use, although, most people prefer to have their executables accessible through their command path.

Setting Windows %PATH%

If you only want to add the *validate.bat* location to your path temporarily, say for testing, you can enter something like this at the command prompt:

```
C:>set PATH=%PATH%;"C:\Program Files\validate-1.11.0\bin"
```

where C:\Program Files\validate-1.11.0\bin should be replaced whatever the full path is to the *bin/* directory containing the *validate.bat* file. This appends the path you provide to the current value of the *%PATH%* environment variable. The double quotes are required if the path you are adding contains embedded blanks.

If you'd like to add the Validate Tool path to your default *%PATH%* once and for all, you can follow the instructions on this page for your particular flavor of Windows:

- How to set the path and environment variables in Windows, by ComputerHope.com.

Setting Linux-based $PATH

The method used for adding a directory to your current *$PATH* varies based on the shell you use. The Bourne shell requires an assignment followed by an export command to make the new path visible to programs you run ('%' is the command prompt in the following examples):

```
% PATH=$PATH:/usr/share/validate-1.11.0/bin
% export PATH
```

or this shortcut should also work:

```
% export PATH=$PATH:/usr/share/validate-1.11.0/bin
```

where /usr/share/validate-1.11.0/bin is replace with the full absolute path to the directory containing the actual *validate* script (not a link).

For C-shell and related shells, use a *setenv* command:

```
% setenv PATH $PATH":/usr/share/validate-1.11.0/bin"
```

or the *set* command:

```
% set PATH=($PATH /usr/share/validate-1.11.0/bin)
```

For either type of shell, you can do this at the command line before beginning you work with *validate*, or you can add the lines to your shell resource file so it's already there every time you open a new shell window.

If you don't know what any of that means, it is time to seek out your friendly, neighborhood Linux programmer and ask, or try Googling "Setting environment variables" for your particular operating system.

Linux Alternative to Extending $PATH: Links

As long as the *validate* script is physically located in the Validate Tool installation tree as described previously, you can create a link to the script from some more convenient place so that you don't have to modify your *$PATH* just to run *validate*. You will need to have write permission to some directory already in your path. You can do this in the *bin/* directory in your own home directory, for example (assuming it is already in your path).

To do this, simply create a link to the *validate* script from the directory already in you path. Say, for example, that the Validate Tool tree is in your home directory and is called "Validate_Tool", so running *ls* on it looks something like this:

```
% ls ~/Validate_Tool
bin   doc   lib   LICENSE.txt   README.txt
```

Create a link to the ~/Validate_Tool/bin/validate *script from your* ~/bin/ *directory thus:*

```
% cd ~/bin
% ln ~/Validate_Tool/bin/validate
```

And that's it. If you want to start using *validate* immediately in the same shell window, you will have to run *source* on your shell resource file (for C-shells that would be source ~/.cshrc, e.g.) to force the shell to re-read you *$PATH* contents. Apart from that rare circumstance, however, *validate* should be in your path every time you start a new shell from now on.

A similar method can be employed by users with sufficient privileges to create a link in an existing shared group or system *bin/* directory for general use.

Mac Users Note

Mac users should be aware of a minor but possibly annoying detail when defining links. The Mac flavor of Linux, while allowing mixed-case file names, does not consider case significant when comparing file names. So if, for example, you decided to install the Validate Tool into *~/bin/Validate*, and then tried to create a link called "validate" to *~/bin/Validate/bin/validate" in the same directory, you'd get an error message telling you a file by that name already exists.*

To get around this you can, of course, move the Validate Tool tree to a different directory; or you can give the link a different name using the second (optional) argument to the *ln* command, e.g.:

```
% ln ~/bin/Validate_Tool/bin/validate pdsval
```

Now, to invoke the *validate* script you would use the *pdsval* link name:

```
% validate --version
```

Test the Installation

Once you think you've got the *Validate* executables tucked into their homes, you should test the installation and configuration.

Aliveness Test

To test whether you can successfully invoke the Java executable, try getting the help listing. This command:

```
validate -h
```

Should produce something like this:

usage: validate <target> <options>	Description
-B,--base-path <path>	Specify a path for the tool to use in order to properly resolve relative file references found in a checksum manifest file
-C,--catalog <catalog files>	Specify catalog files to use during validation.
-c,--config <file>	Specify a configuration file to set the tool behavior.
-e,--regexp <patterns>	Specify file patterns to look for when validating a directory. Each pattern should be surrounded by quotes. Default is to look for files ending with a '.xml' or '.XML' file extension.
-f,--force	Force the tool to perform validation against the schema and schematron specified in a given label.
-h,--help	Display usage.
-i,--integrity-check	Perform referential integrity checking on the given target directory or directories.
-L,--local	Validate files only in the target directory rather than recursively traversing down the subdirectories.
-M,--checksum-manifest <file>	Specify a checksum manifest file to perform checksum validation against the targets being validated.
-m,--model-version <version>	Specify a model version to use during validation. The default is to use the latest model.

-r,--report-file <file name>	Specify the report file name. Default is standard out.
-S,--schematron <schematron files>	Specify schematron files.
-s,--report-style <full|json|xml>	Specify the level of detail for the reporting. Valid values are 'full' for a full view, 'json' for a json view, and 'xml' for an XML view. Default is to see a full report if this flag is not specified.
-t,--target <files,dirs>	Explicitly specify the targets (files, directories) to validate. Targets can be specified implicitly as well. (example: validate product.xml)
-v,--verbose <1|2|3>	Specify the severity level and above to include in the human-readable report: (1=Info, 2=Warning, 3=Error). Default is Warning and above.
-V,--version	Display application version.
-x,--schema <schema files>	Specify schema files.

Alternately, you can view the version information for the executable:

```
validate --version
```

(*--version* can be abbreviated as *-V*, but note the capital. There is also a lower-case *-v* verbosity option.) This will produce a listing something like this:

```
Validate Tool
Version 1.11.0
Release Date: 2017-04-03 11:57:19
Core Schema: PDS4_PDS_1800.xsd
Core Schematron: PDS4_PDS_1800.sch

Copyright 2010-2017, by the California Institute of Technology.
ALL RIGHTS RESERVED. United States Government Sponsorship acknowledged.
Any commercial use must be negotiated with the Office of Technology Transfer
at the California Institute of Technology.

This software is subject to U. S. export control laws and regulations
(22 C.F.R. 120-130 and 15 C.F.R. 730-774). To the extent that the software
is subject to U.S. export control laws and regulations, the recipient has
the responsibility to obtain export licenses or other export authority as
may be required before exporting such information to foreign countries or
```

```
        providing access to foreign nationals.
```

Anything else indicates a configuration error of some sort. Re-check your paths and script/batch file editing and try again. If you can't resolve the problem yourself, contact your local PDS consultant for additional assistance. Please provide the failing file(s) and as much detail as possible.