Using Validate Tool

The *Validate* tool is the canonical PDS4 validator for single labels, collections, and bundles. It provides additional functionality beyond schematic development, and advanced features are still in active development.

If you have not already installed and configured *Validate*, you will find instructions on the Installing and Configuring Validate Tool page, as well as included in the download bundle.

The following information was updated for *validate* version 1.11.1.

Format

The general command format is:

`% validate [files] [options]`

"*Files*" can be a single label file name, a directory, or a comma-separated list of both. This list may include absolute and relative paths, wild cards, and file globs. Alternately, the product labels to be validated can be specified as arguments to the **-t** option.

Default Behavior

- If *validate* is given a directory as an argument it will attempt to validate all files ending in either ".xml" or ".XML" with the selected options and will recurse down through all subdirectories doing the same.
- For each label, *validate* will:
- Check conformance to the schema and Schematron files referenced or supplied;
- Confirm that all referenced data files exist (note that case in file names is significant)
- Calculates and compares data file MD5 checksums to values provided in the corresponding label via the *<md5_checksum>* attribute; and
- If an MD5 checksum manifest file is provided (via the **-M** option), calculates MD5 checksums for the label and data files and compares them to the values in the manifest file.
- For each directory, *validate* will run label validation on every file it finds that ends in ".xml" or ".XML", and recurse down through subdirectories doing the same.
- When the *validate* argument list contains both individual labels and directories, the default label validation is applied to all labels, and the default directory validation is applied to all directories.

Command Line Options

These options have been grouped by functional areas. All options have both a short and a long form. One or two hyphens can be used for both short and long options interchangeably; long options *cannot* be truncated. Most options require arguments, but even those that do not can *not* be globbed. That is, specifying "-*Vh*" will display only version information, not help information.

Alternately, a configuration file can be used to give values for most of the following options. The Validate Tool Configuration File page provides details.

Program Information

These options provide information about the program itself.

Option	Argument	Notes
-V --version	*--none--*	This option displays the internal code version, the release date, and the core schemas applied by default by this version of *validate*, in addition to some licensing boilerplate.
-h --help	*--none--*	This option displays a command and option summary. It is about 50 lines long, so prepare to scroll.

Selecting What to Validate

These options, along with any files and directories included in the argument list, select and refine the file and directories validated. Whether included as arguments to *validate* or arguments to the **-t** option (below), the *Validate Tool* documentation refers to these collectively as "targets".

Note: *No merging or duplicate removal is done on the list of files and directories supplied. If the list, when expanded, includes the same file or directory more than once, that element will be validated* in its entirety *each time it is listed. This is a waste of time - potentially a significant one for large file collections - so type carefully.*

Also, be warned that the *Validate Tool - Operation* documentation included with the tool uses the word "implicit" to mean "explicit", and consistently mis-uses both terms in an attempt to distinguish between targets specified as *command arguments* and targets specified as *option arguments* (i.e., arguments to command options, like **-t**). The examples in the document do reflect actual program behavior.

Option	Argument	Notes
-t --target	*file/directory list*	Use this option as an alternate way to specify which files and/or directories (i.e., "targets") to validate. In the absence of the **-t** option, the comma-separated list of targets *must* immediately follow the *validate* command. By using the option, the target list can appear at any point among the other options. The syntax and globbing options are identical whether the **-t** switch is used or not. It is possible to provide two target lists: one with the **-t** option, and one without. In this case the two lists are concatenated, and any options that modify file selection apply to the merged list.
-L --local	*--none--*	Note the uppercase "L" in the short version of this option. When present, this option prevents *validate* from recursing down into subdirectories of any directories included in the argument list or **-t** target list.

-e --regexp	*pattern[, pattern]*	This option appears to be misnamed. It does not accept regular expressions in general, but rather allows an addition option for file-globbing beyond what might be in the argument or target list. It is applied only to the file name, not the path, and behaves as if anchored to the beginning and end of the file name. So **-e "*.XML"** will select only files with the explicit uppercase "XML" extension. Beware of selecting files by name without extension. The option **-e "table*"** will select all files beginning with "table" regardless of file extension - most likely producing a validation error when *validate* attempts to validate data files as well as labels.

Output Control

These options affect the format and destination of the output report. Three "INFO" messages produced at the start of every *validate* run go to the standard error output, but everything else is directed to standard output unless this option is present. You can also redirect the output to a file or a process, if desired.

Option	**Argument**	**Notes**
-r --report-file	*file name*	This option directs the output to the named file.
-v --verbose	**1 | 2 | 3**	This option affects the "Validation Details" listing in the report by changing the severity level of the messages generated. A value of **1** causes *INFO* level and above messages to be included in the details; a value of **2** (the default) includes *WARNING* level and above; a value of **3** includes only *ERROR* level messages. Note that the *PASS* and *FAIL* messages are always generated, irrespective of the verbosity setting.
-s --report-style	**full | json | xml**	The default report format, corresponding to the **full** value, is human-readable. This option allows you specify an alternate form for the output that is more congenial to programmatic analysis. Two alternatives are currently available: *json*, which produces JSON output; and *xml*, which produces output with XML tags (there is no schema published for these tags, but the tag names are formulated from the titles present in the default report format). Note that these values must be in lowercase. Using "JSON" rather than "json", for example, will not produce the expected error message for an unrecognized value, but rather just the literal word "null".

Specifying Schemas

The *Validate Tool* comes with the released schema and Schematron files for the core ("pds") namespace built-in. By default, it will use the latest version of these for validation. If you are not sure what that version is, use the **-version** option to check. Older releases of the core namespace can be indicated via the **-model-version** option (below).

All other dictionary schema/Schematron files, including discipline dictionaries and any local dictionaries you have created, must be read in by *validate* - so you will have to have access to the relevant schema/Schematron files and will have to direct *validate* where to find them. A number of options are provided for that.

Note: *PDS4 dictionaries consist of two files: a schema (.xsd) file and a Schematron (.sch) file. Both are needed to fully define any PDS4 namespace (core, discipline, or local). If you are validating files that reference non-core namespaces and do not provide the schema file, you will get an "ERROR" message in the "Validation Details" list containing a phrase like* "The matching wildcard is strict, but no declaration can be found for element..."*, followed by an element from the missing namespace. If you omit the Schematron file, however, no notice of any kind will be generated -* validate *cannot tell that a Schematron file is missing due to the nature of Schematron validation (i.e., further constraints on top of the definitions contained in the schema file).*

Triple-check your Schematron file lists to avoid missing validation errors.

Option	**Argument**	**Notes**
-m --model-version (DEPRECATED)	*version code*	By default, *validate* will validate all labels against the default version of the core namespace (listed in the **-version** option output). You can change the core namespace version used for validation to any previously released version with this option. The "version code" is the version ID with all the '.' characters removed - so to validate all labels against the 1.7.0.0 version of the core namespace, use "*-m 1700*". **Note:** *Use caution when combining this option with the* ***-x*** *and* ***-S*** *options (see below). This option is ignored without comment when the* ***-f*** *option is present.*

-f --force (DEPRECATED)	*--none--*	This option forces *validate* to validate each label against the schema and Schematron files actually referenced in the label (via, for example, *schemaLocation* attributes). Those references *must* be resolvable. So, for example, you use a *schemaLocation* style that looks like a URL, *validate* will attempt to connect to that URL to download the file. If it can't, the program will issue a "FATAL_ERROR" and move on to the next label. **Note:** *This option* ***cannot*** *be used in conjunction with the* **-catalog** *option, so labels may contain only absolute paths or URL references that can be resolved through a network connection for this option to be viable. Neither can this option be used with the* **-schema** *and* **-schematron'** *options.*
-C --catalog	*XML catalog file*	This option takes an XML catalog file as an option and attempts to use it to resolve public and system ID references for validation files. (See the Understanding XML Catalog Files page for more information on catalog files and their use.) In most respects, it fails. There is an example of a catalog file that might work in the *Validate Tool* Operations guide, but the same effect can be achieved with less effort by using a configuration file (see below) with the schema file lists. In particular, this option fails to properly apply an XML Catalog file that makes use of the *<rewriteURI>* element. This, in conjunction with the fact that it cannot be combined with the **-force** option, where you would likely want to be able to locate multiple different versions of various schema files, makes the functional usefulness of this option extremely limited.

-x —xsd	*XSD file list*	Use this option to list the schema (i.e., the "*.xsd") files, by location, to be used for validating labels. Typically, this will be used to provide the XSD part of discipline and local dictionary schemas, and must be used in conjunction with the **-schematron** option. If you are using an older version of *Validate* with a new version of the core schema (a version later than the default version built into the tool), you will also need to supply that schema location via this option. The "location" can be either a directory on your system, or a URL (if you have a network connection and valid URL). ***Caveats:*** • This option cannot be used in conjunction with the **-force** option. • If you are listing a core schema file, it must come *first*. • If you list more than one version of any single dictionary XSD file, only the last one listed will be applied. • Using this option more than once is not an error. The program behaves as if the various arguments to the option were concatenated into a single, comma-separated list in the order specified on the command line. • Using this option without the **-schematron** option is not an error and doesn't generate a warning - even though every PDS4 dictionary that has a schema file also has a Schematron file that must be used for validation as well. • Listing an XSD file in this option **does not** automatically include the corresponding Schematron (.sch) file - you *must* include it explicitly via the **-schematron** option. • There is no attempt to match the XSD file list in this option with any Schematron list that might or might not be provided via the **-schematron** option. • Attempting to use this option with the **-model-version** option does not produce an error, but the **-model-version** *will* override the core schema version listed by this option (but see the similar note for the **-schematron** option).

-S --schematron	*SCH file list*	Use this option to list the Schematron (i.e., the ".sch") files, by location, to be used for validating labels. Typically, this option will be used to provide the Schematron part of discipline and local dictionary schemas, and must be used in conjunction with the **-schema** option. If you are using an older version of *Validate* with a new version of the core schema, you will also need to supply that Schematron location via this option. The "location" can be either a directory on your system, or a URL (if you have a network connection and valid URL). ***Caveats:*** • This option cannot be used in conjunction with the **-force** option. • If you list more than one version of any single dictionary Schematron file, **both** will be applied to all labels. This may result in either duplicate or conflicting errors flagged by the Schematron files. • Using this option more than once is not an error. The program behaves as if the various arguments to the options were concatenated into a single, comma-separated list in the order specified on the command line. • Using this option without the **-schema** option is not an error and doesn't generate a warning - even though every PDS4 dictionary that has a Schematron file also has a schema file that must be used for validation. • Listing a Schematron file in this option *does not* automatically include the corresponding schema file. • There is no attempt to match the Schematron file list in this option with any schema files listed via the **-schema** option. • Attempting to use this option with the **-model-version** option does not produce an error, but - *unlike the case with the **-schema** option* - this option will override the **-model-version** Schematron.

Additional Validation

These options direct the *Validate Tool* to perform validation steps beyond the schema validation.

Option	Argument	Notes

-R --rule	**pds4.label** **pds4.folder** **pds4.collection** **pds4.bundle** **pds3.volume**	This option is the only way to get referential integrity checking on PDS4 collection and bundle membership. The **pds4.label** and **pds4.folder** values correspond to the default behavior for individual labels and directories. The **pds3.volume** option is provided for validating PDS3 volumes being delivered by grandfathered pipelines and thus will not be discussed further here. Note that it is not an error to specify this option more than once, but only the last value specified will have any effect. • There is no reason to ever specify **pds4.label**. It is the default for single labels anyway, and specifying it for a directory is a fatal error. • Similarly, there is no reason to ever specify **pds.folder**. It is the default for directories, and specifying it for a single label is a fatal error. It is also the baseline for validation of collection and bundles. • Specifying **pds4.collection** for a collection directory (*not* the collection label) performs referential integrity on the collection inventory list to ensure all member products are present and accounted for, and also checks for restricted and prohibited file names as specified in Section 6C of the Standards Reference. • Specifying **pds4.bundle** for a bundle directory (*not* the bundle label) applies the **pds4.collection** validation to all subdirectories and checks referential integrity for bundle members.
Note: *In addition to the checks listed, the **pds4.collection** and **pds4.bundle** values also cause* validate *to flag as errors file and directory names that do not conform to the* optional *specifications in section 2B of the* Standards Reference*. You should confirm delivery formats and naming conventions with the PDS node receiving your data prior to delivery (typically as part of your Data Management Plan or Archive Plan review). If you opt to use different conventions than the ones listed in section 2B, the "errors" flagged with respect to required file and directory names from this section can be ignored.*		

-M --checksum-manifest	*checksum file*	Use this option to provide a list of MD5 checksums for integrity checking of label and data files. The file should consist only of checksum lines with the MD5 checksum first and the file name (with path) following. This is the format output by, for example, the linux *md5sum* routine. Typically, path names in checksum files are relative to the root directory passed to the checksum generation program. If the paths in the file are relative to the root of the collection or bundle, then this option alone will likely be sufficient. If you encounter multiple errors saying *"No checksum found in manifest"* for files that should have been found, then you will likely need to use the **-base-path** option (following) to provide explicit path information.
-B --base-path	*absolute path*	Use this option to provide an absolute path to prepend to the file specifications in the checksum manifest file to resolve reference issues locating the files. This option has no effect if the **-checksum-manifest** option is not also present.

Configuration File

You can create a configuration file to specify options rather than (or in addition to) supplying them on the command line. This is particularly useful for providing schema locations, as you might otherwise do with the **-schema** and **-schematron** options. Note that options provided on the command line will override options in the configuration file ***if and only if*** the **-c** option comes *before* the command line option intended to override it.

The format and content of the configuration file is described on the **Validate Tool Configuration File** page.

Option	**Argument**	**Notes**
-c --config	*config file*	Use this option to specify a file containing configuration directives for running *validate*. You can include path information in the file specification if needed.

Note: *It is an error to try to list multiple configuration files for this option, but it is* not *an error to use the* ***-config*** *option multiple times. When you do that, the options from the two files are combined - somehow. This is an off-label use of configuration files that should probably be avoided.*

Common Error Messages

These are error messages generated by the tool itself, as opposed to the validation errors the tool might report.

null

If the only word in your output listing or report file is the string "null", you've got an error in your command line somewhere. Check spelling (no truncation of long-form options is allowed) and case of options and their arguments to find the culprit.

ERROR Uncaught exception while validating: Input file is not a directory: ...

This is usually caused by a typo in your command (or configuration file). Look at the name of the "file" for a clue. This can be caused by a mistyped long option as well as mistyped file or directory names.

_ERROR line [_number, character_]: src-resolved: Cannot resolve the name '[_name_]' to a(n) 'element declaration' component_

There are some variations on this theme, but the "Cannot resolve" message means that a schema (XSD) file you attempted to reference directly via command line or configuration file options, or perhaps indirectly through a label *schemaLocation* when using the **-force** option, could not be found and read. Check for typos in the options, or in the case of **-force**, the label file that generated the error.

ERROR No checksum found in the manifest for 'file:...'

This message results when *validate* cannot find a file listed in the checksum manifest. If you are not using the **-base-path** option, you may need to. If you are, check it for typos, and for overlapping path elements with what is in the manifest file filenames.