Installing and Configuring LDDTool

A *local data dictionary* is a set of schema files that define a namespace that is under the control of someone other than the PDS4 managers. It includes the PDS discipline dictionaries for things like display orientation and geometry, as well as node- and mission-specific dictionaries. There are web-based and GUI-based tools in development at various places to help users who prefer to do dictionary development in a web/GUI environment - ask your friendly, neighborhood PDS node consultant what's currently available if that's what you are looking for. These pages are for the roll-your-own crowd that either prefers or has no choice but to work at the command line and see how the sausage is made.

Caveat Usor

Be advised: There is a fair amount of hands-on setup work required to get the LDDTool working in your local environment the way you want it to. And because this tool is updated for each new model release and to accommodate the sometimes complex needs of the discipline dictionaries still in development, you may well have to repeat this process with some variations with each new release. We'll try to keep this page updated to reflect the latest version of the tool. Feel free to add additional information about LDDTool versions or OS versions not specifically mentioned here.

Goal

Our goal in this set of pages is to start with the LDDTool 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 looks something like this:

```
% lddtool -lp <input_file>
```

Part List

To run the LDDTool locally, you'll need the following:

- **The LDDTool package**. The package is available as either a ZIP file or a compressed TAR file from the PDS4 Local Data Dictionary Tool GitHub page, https://nasa-pds.github.io/pds4-information-model/model-lddtool/. Either format will do.
- **Java 1.7 or later**. Type "java -version" at your command line to see 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 1.7 and later includes a handy feature that will help with configuration later on, so if you are still running an older 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 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-Windows7), or *Notepad* (which can be invoked from the Windows command line as *notepad) on newer ones.*
- **An XML editor**, while optional, will make editing the output schema files easier, and you'll probably want one for creating the input file anyway. A schema-aware editor like *Eclipse* (open source) or *oXygen* (commercial) can be handy for one-off file creation and editing. For the minor fix-up editing needed in the LDDTool output schemas, though, you can use the same simple text editor you used to edit the batch file or command wrapper.

General Procedure

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

1. Unzip the LDDTool package.
2. Move the directories you actually 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 with the supplied sample files.
6. Rejoice in the knowledge of a job well done.

Procedure

Unzip the LDDTool Package

Use any 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. You should end up with a directory with a name that starts with *lddtool-* and ends with the version number of the tool. As of this writing, the latest version of the tool is 8.0.0, so the delivery package unpacks into a directory called *lddtool-8.0.0*. On a Windows system, by default this directory will be underneath a directory with the same name as the ZIP package, less the ".zip" extension. You can unpack it anywhere - we'll move the stuff we need to a new home once we've picked one out. If you haven't inspected previous *LDDTool* 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 actually only need one of the files from this directory, but you'll have to carry the directory along nonetheless. We'll be modifying one of these scripts to work on your local system, and then installing just that modified file into an *LDDTool*-specific `bin/` directory.
The *runapp/* subdirectory contains some batch files that, depending on your system, could be used to run the tool on some of the example input files provided.
Data/
At least one of these files is referenced directly by the LDDTool java code, so it needs to be present.
lib/
This directory contains only the `DMDocument.jar` jar file, which contains the actual Java 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. There are brief summary instructions for installation and usage in the nominal case.

Examples

This directory contains sets of input/output files produced by *LDDTool*. They are undocumented, but at least if you run the usual *lddtool -lp* command on the *IngestLDDTool_*.xml* you should get output

similar to the file set here. We'll use one of these for testing the installation. Apart from that, they may be useful examples for how to code some of the more specific, more complex behaviour found in the more intricate discipline dictionaries.

Peanuts

Like packing peanuts, these files are included in the ZIP but are not, as far as I have found, particularly useful once the package is opened:

- *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.
- *bin/runapp.bat*: A Windows-style batch file that demonstrates how to invoke an executable in the bin directory for a (non-existent) sample file *IngestLDDTool.xml*. It will not run as is if you try to execute it.
- *bin/runapp/*: This directory contains a variety of batch files that look like they should be able to run LDDTool on some of the example files provided. They will not succeed in their current location and configuration, but they provide some minimal guidance on command line variations.
- *Schemas/*: This directory contains copies of the last few releases of schema files for the core PDS4 namespace. I'm not sure why it's here; the tool seems to run fine if this directory is expunged. Could be handy if you don't already have a schema tree but want to use a schema-aware editor to create your *LDDTool* input file.

Install the Executable and Support Directories

Unless you are seriously hardcore, you will be running LDDTool by invoking a wrapper script (or batch file). This script sets up some environment variables and then calls Java with the appropriate options and arguments to invoke the *DMDocument.jar* file with the options and arguments passed on from the wrapper script.

Note: The classes in the *DMDocument.jar* file read all the environment variables set by the wrapper script/batch file, and also contain hard-coded references to the Data subdirectory in the installation tree. So wherever you install LDDTool, you are going to need to preserve the delivery tree structure for the bin/, lib/, and Data/ subdirectories - and the wrapper script *must* be physically located in that bin/ directory.

Choosing an Installation Location

On linux-based multi-user systems, you can install LDDTool for general use by all users either by installing into one of the standard locations (/usr/share, for example), or in shared disk space. If the latter, users wanting to execute LDDTool will likely have to add the appropriate location to their $PATH setting. Alternately, you can install it into your own ~/bin/ directory for personal use. 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 system you will ultimately have to choose one of these options:

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

On Windows systems, you can install LDDTool into the "Program Files\" directory for general use (this may require admin privileges), or in your own directory space for personal use. You will likely have to modify %PATH% setting information to make the lddtool.bat executable visible to users without requiring a complete path specification to run the batch file. More on that later.

What to Copy/Move

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

Under this directory, copy over the entire contents of the lib/ and Data/ directories from the installation package. You will also need to create a bin/ directory, into which you should copy either the lddtool linux script or the lddtool.bat Windows batch file, as appropriate for your environment. For linux users, you will likely also have to make the lddtool script executable.

At this point you may also want to copy over the contents of the doc/ directory, for easy reference. I also copy the *README.TXT* and *LICENSE.TXT* files from the root of the install package into this directory, just in case I want to find them again later.

Edit the Wrapper Script/Batch File

The lddtool script (linux) or lddtool.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 can do the job.

Windows Batch File *lddtool.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 something.

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

```
SETLOCAL
```

This makes sure that any variables that are set by this script 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'll see the (uncommented) lines that check whether the *%JAVA_HOME%* environment variable is already set, and if it isn't, sets it. See the Finding and Setting JAVA_HOME page for detailed steps to check the variable and find the right value to insert here if the variable is not already set.

The last executable line in the script before the :END statement looks like this:

```
"%JAVA_HOME%"\bin\java -jar "%DMDOC_JAR%" %*
```

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

Paths with embedded blanks and extra sets of quotes can 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 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 *lddtool* script

If your $JAVA_HOME environment variable is not already set, you will need to edit the export JAVA_HOME line to set it. See the Finding and Setting JAVA_HOME page for details.

Note: The *lddtool* wrapper 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.

If you are planning to add lddtool to an existing bin/ directory (as opposed to adding a new element to your *$PATH* to access the tool) you'll need to edit one additional line in the lddtool wrapper - the line beginning export PARENT_DIR (line 34 in the current distribution). Replace the back ticks (`) and everything inside them with the absolute path to the LDDTool installation directory (*without* ticks or quotes).

Now make the lddtool script file executable, and you are ready to test it.

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 (Windows or Linux) do:

```
lddtool -v
```

The response should look like this:

```
LDDTOOL version: 0.2.1.0
```

Yes, this version number here is different from the version number of the package. The version number above corresponds to the version 7.0.1 release package.

Once you have had some experience with running LDDTool and tried out some of the other options available, you may want to further modify the script or batch file to automatically include certain options, provide a standard output file redirect, and otherwise customize tool behavior.

Install the Wrapper Script/Batch File

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

You can, of course, always execute the script/batch file by using its full, absolute path on the command line. For ease of use, though, most people prefer to have their executables available in their path.

Setting Windows %PATH%

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

```
C:>set PATH=%PATH%;C:\Users\LDDTool\bin
```

where C:\Users\LDDTool\bin should be replaced with whatever the full path is to your LDDTool *bin/* directory. This appends the path you provide to the current value of the *%PATH%* variable. You will need to use double quotes around the path you are adding if it contains embedded blanks.

If you'd like to add the LDDTool path to your default *%PATH%* once and for all, you can follow the instructions on this page for your particular version 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:

```
% PATH=$PATH:/usr/share/LDDTool/bin
% export PATH
```

or this shortcut should also work:

```
% export PATH=$PATH:/usr/share/LDDTool/bin
```

where `/usr/share/LDDTool/bin` is replaced with the full path to the LDDTool installation tree *bin/* directory.

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

```
% setenv PATH $PATH":/usr/share/LDDTool/bin"
```

or the `set` command:

```
% set PATH=($PATH /usr/share/LDDTool/bin)
```

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

If you don't know what any of this 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

So far, at least, as long as the *lddtool* script is physically located in the LDDTool 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 *lddtool*. You'll need to have write permission to some directory already in your path. You can do this in your own *~/bin/* directory, for example (assuming it's already in your path).

To do this, simply create a link to the *lddtool* script from the directory already in your path. Say, for example, that the LDDTool tree is in your home directory and is called `LDDTool`:

```
% ls ~/LDDTool
bin         Data        doc         lib
```

Create a link to the *~/LDDTool/bin/lddtool* file in the *~/bin/* directory thus:

```
% cd ~/bin
% ln ~/LDDtool/bin/lddtool
```

If you want to start using *lddtool* immediately in the same shell window, you will have to `source` your shell resource file to force it to re-read your path contents. Apart from that rare circumstance, *lddtool* 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 system *bin/* directory for general use.

Mac Users Note

Mac users should be aware of a minor but possibly annoying detail when defining aliases. The Mac version 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 LDDTool into *~/bin/LDDTool*, and then tried to create a link called "lddtool" to *~/bin/LDDtool/bin/lddtool* 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 LDDTool tree; or you can give the link a different name using the second argument to the `ln` command:

```
% ln ~/bin/LDDTool/bin/lddtool makeldd
```

Now to invoke the *lddtool* script, you would use the *makeldd* alias, e.g.:

```
% makeldd -lpM IngestLDDtool.xml
```

I haven't actually tested whether or not you can rename the *LDDTool* directory itself. If you do try that and have anything to report, let me know...

Test the Installation

Once you think you've got the LDDTool executables in their correct locations, you should test the installation and configuration. You can use the File:LDDTool 701 examples.zip for testing if you don't have an input file of your own yet.

Aliveness Test

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

`lddtool -h`

Should produce something like this:

```
Usage: lddtool -pl [OPTION]... FILE1 FILE2 ...
Parse a local data dictionary definition file and generate PDS4 data standard files.

Example: lddtool -pl  inputFileName

Process control:
  -p, --PDS4      Set the context to PDS4
  -l, --LDD       Process a local data dictionary input file
  -a, --attribute Write definitions for attribute elements.
  -c, --class     Write definitions for class elements.
  -J, --JSON      Write the master data dictionary to a JSON formatted file.
```

```
  -m, --merge     Generate file to merge the local dictionary into the master dictionary
  -M, --Mission   Indicates mission level governance (includes msn directory specification)
  -n, --nuance    Write nuance property maps to LDD schema annotation in JASON
  -s, --sync      Use local namespace + information model version as output file names.
  -1, --IM Spec   Write the Information Model Specification with LDD.
  -v, --version   Returns the LDDTool version number
  -h, --help      Print this message

Input control:
  FILEn provides the file name of an input file. The file name extension .xml is assumed.
    If there are more than one file, the first files are considered references
    for the last file. The last file is considered the primary local data dictionary.

Output control:
  FILE is used to provide the file name for the output files. The file name extensions are distinct.
  .xsd -- XML Schema file
  .sch -- schematron file
  .xml -- label file
  .csv -- data dictionary information in csv formatted file.
  .JSON -- dump of model in JSON format.
  .txt -- process report in text format
  .pont -- ontology file for merge
```

Note: *There is more wrong than right in the* "Process Information" *section displayed. Ignore it and use the information provided by the Running LDDTool and Verifying the Output page instead.*

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

```
lddtool -v
```

which should produce something like this:

```
LDDTOOL Version: 0.2.1.3
```

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.

Running LDDTool on the Example Files

If you haven't already, download the LDDTool 1900 examples.zip package, and unzip it into a working directory. This package contains some example *Ingest_LDD* input files that demonstrate dictionary creation techniques. The package contains output files from running *lddtool* to generate the included schemas - you probably want to stow those somewhere to compare them to what you are about to create.

The input dictionary files are called "IngestLDD_Example_Attributes.xml" and "IngestLDD_Example_Classes.xml". To invoke *lddtool* to duplicate the output files included in the package, do:

```
lddtool -lpM IngestLDD_Example_Attributes.xml > IngestLDD_Attributes.out
```

and/or:

```
lddtool -lpM IngestLDD_Example_Classes.xml > IngestLDD_Classes.out
```

The commands above redirect the information that would normally scroll by on your screen to the *.out* file so you can examine it at your leisure and compare it to the version provided in the package. It is also where errors detected by *lddtool* are reported.

Expected Results

The example *lddtool* command above will generate a total of five output files in addition to the *.out* listing file. The files will all have the same *lddtool*-generted names but different extensions. Here are those extensions, in approximate order of usefulness:

- **.xsd**: This is the XML Schema file that you will reference in your labels when you want to use classes from this dictionary.
- **.sch**: This is the Schematron file that you will also reference in your labels when you want to use classes from this dictionary.
- **.csv**: This is a CSV-formatted summary of the dictionary contents. You might find this a useful way to review the results if you are averse to reading schema and don't have labels already written to exercise the newly-produced schemas. You might also find this to be a useful file for passing to reviewers who want to see class and attribute definitions - though maybe with a little editing first.
- **.xml**: This is a label for the XML Schema and Schematron files; probably only useful as a template. We strongly recommend that rather than creating a label from scratch each time, you modify an existing label at reasonable intervals in order to maintain a `<Modification_History>` within the label that accurately reflects the development history of the dictionary (as any other product label should for an archival product). At least, the schema label should be modified to identify the unique origin and application of the dictionary files it describes.

Checking for Success

If you compare the output files from the above commands to what came in the example zip file, you should see differences in date stamps and local paths, but otherwise nothing else. Make sure your option string is **-lpM** (lowercase letter "ell", uppercase letter "em"), or you will get a slightly different set of output files or a different namespace definition in the output schemas.

If you run the tool on your own input file, the first thing to check is the program output listing, which will scroll past on your screen if you don't redirect it to a file. The last line of that listing should look like this:

```
    >>info    - LDDTOOL Exit
```

This indicates a measure of success. Depending on how complex your input file is, there will be a few dozen to a few hundred "INFO" lines containing messages about various override conditions. This is normal. It should *not* contain any "ERROR" lines or lines beginning with ">>error". These indicate some sort of failure. There will likely also be two "WARNING" lines that look like this:

```
    WARNING  Header:  - New steward has been specified:sbn
    WARNING  Header:  - New namespace id has been specified:ex
```

unless you are updating a dictionary that is already known to LDDTool. Other "WARNING" statements, however, are problematic and should be investigated.

Once you've verified expected output, you should be good to go.

Common Failures

The common failures encountered at this point come from system references not resolving. Here are the most likely suspects:

Cannot find DMDocument jar file in [*some directory****]***

This error is reported back to the command line by the lddtool wrapper, which checks for the existence of the *DMDocument.jar* file before invoking java on it. If you haven't previously set PARENT_DIR in the wrapper to point to the LDDTool installation directory, do so (in some environments this may be required even if you are using the default configuration). If you have already modified PARENT_DIR, search it for typos. The PARENT_DIR directory *must* contain a lib/ subdirectory, which in turn must contain the DMDocument.jar file.
You'll also get this message if there is a typo in the lib/ subdirectory name or DMDocument.jar name (case counts).

>>error - Required data file was not found: [*some path to an XML file****]***

This will show up as the last line in your listing if you forgot to include the Data/ subdirectory of the *LDDTool* distribution in your installation LDDTool directory tree. Spelling and case count, so if you are on a linux-based system and accidentally changed "Data" to "data", for example, you'll get this message.

/bin/java: No such file or directory (This is the linux version of this error)

Messages like this are reported to the command line and indicate that the JAVA_HOME setting either failed or points to the wrong place. If there's a typo in the JAVA_HOME setting, you might also see characters before "/bin/java" indicating what the script thinks JAVA_HOME was set to.
The JAVA_HOME directory *must* contain a bin/ subdirectory, which in turn must contain the *java* executable.

[*usage info dump****]***

If you get a dump of *lddtool* usage information when you run the test command on the example file, look at the top - there's probably an error waiting for you. Make sure you spelled the input file name correctly and included the required "-lp" option set.