

LAND PEATE CHECKLIST

GENERAL ORGANIZATION

The NPP2 repository is constructed differently from the NPP repository. Principal differences include

1. Library organization for build 48 related code:

- a. All PGEs, shared codes and libraries may be built using flags for MODIS or VIIRS tile sizes (VIIRS tile size is the default) and may be built in DEBUG mode (Production mode, which does not use the debug flag, is the default)
- b. The top-level NPP2 directory now contains library subdirectories containing built 32- and 64-bit libraries which may be downloaded (NPP2/lib32/VIIRS_TILE_SIZE, NPP2/lib32/MODIS_TILE_SIZE, NPP2/lib64/MODIS_TILE_SIZE, NPP2/lib64/VIIRS_TILE_SIZE.) These libraries are static libraries and are production mode.
- c. The directory NPP2/Library/ contains subdirectories to contain DEBUG versions of the libraries. Unlike the production versions, the debug versions are not entered into the NPP2 repository and must be built by the user. Some debug library versions are dynamic.
- d. The directory NPP2/Library/buildfiles contains master build files and makefiles to build current versions of all the libraries in any of the 4 possible modes through the use of command-line directives. The buildfiles directory also contains subdirectories which contain buildfiles specific to each of the four instances.
- e. Libraries are organized into 5 groups. You should read the document README.Library.txt file located in the NPP2/Library/COMB directory to familiarize yourself with these groups.

2. Library organization for build Mx4 (IDPS v1.5.04.00) related code:

- a. All OPS PGEs, shared codes and libraries may be built in either STANDARD or DEBUG mode. By default, the Land PEATE builds the codes to use the MODIS_TILE_SIZE for the OPS Gridding codes.
- b. The top-level NPP2 directory for Mx4 is NPP2/branches/Mx4 which contains library subdirectories that store built 32- and 64-bit libraries which may be downloaded (NPP2/branches/Mx4/lib32/STANDARD, NPP2/lib64/STANDARD.) These libraries are static libraries and are production mode.
- c. The directory NPP2/branches/Mx4/Library/ contains a subdirectory labeled "DEBUG" which is where the DEBUG versions of the libraries are stored. However, unlike the production versions, the debug versions are not entered into the NPP2 repository and must be built by the user. Some debug library versions are dynamic.
- d. The directory NPP2/ branches/Mx4/Library/buildfiles contains master build files and makefiles to build current versions of all the libraries in any of the 4 possible modes through the use of command-line directives. The

buildfiles directory also contains subdirectories which contain buildfiles specific to each of the four instances.

- e. Libraries are organized into 5 groups. You should read the document README.Library.txt file located in the NPP2/branches/Mx4/Library/COMB directory to familiarize yourself with these groups.

3. Top Level directory organization

The top-level build 48 directory, NPP2, or for Mx4, NPP2/branches/Mx4, contains several subdirectories:

- a. The “lib32”, “lib64”, and “Library” directories mentioned above.
- b. A “Templates” directory which contains header templates for code, makefiles, and buildfiles.
- c. A “Tools” directory which contains various off-line programs which may be built and / or run by developers or users.
- d. A “Config” directory containing the master setup files to set up the user environment.
- e. A “Documentation” directory containing Land PEATE documentation such as the System Description Document.
- f. A “SCIENCE” subdirectory containing Science code and also containing the Land PEATE DDRs adapted from MODIS code.
- g. An “OPS” subdirectory which contains the LPEATE PGEs, Libraries, and Include files.
- h. An “IDPS” subdirectory which mimics the structure of the Operational code developed by Raytheon/NGST. *The code in this directory has been adapted from the original code.*

PHILOSOPHY

Whenever we as developers make adaptations to the IDPS code, we want to do so in a way that is as non-intrusive as possible. (That is unfortunately not always something that can be done, but changes are kept to a minimum). At the same time we want absolutely no dependence on the IDPS Dynamic Memory System (DMS) module and you will note that it is not a subdirectory of the IDPS directory (the only subdirectories in subversion’s IDPS are PRO, where the bulk of the algorithm code resides, INF, which contains some informative/control structures but is extensively gutted from the IDPS original, and the ING directory, which contains definitions of various configuration-defining structures).

To this end,

- If your PGE depends on IDPS libraries, and it obviously will, try to understand those dependencies.
- If your code needs include files which are not in the ../include directory where your algorithm resides, copy them into the locations in the repository’s IDPS directory that correspond to their location in the original IDPS code (and look to

be sure they aren't there already before you do so.) You may need to edit them somewhat but be sure to keep it to a minimum.

BEFORE YOU START A NEW CODE INTEGRATION

1. Locate the new code in the code delivery area.
2. Look at the XML config file for that module in PRO/cfg. Are there any changes to it from the previous version?
3. Look at the LUTs which are required for input to your module. Are they identical to those for the previous version? If so, you're in luck. You can continue to use the already byteswapped versions. If they are not, don't forget that they must be byteswapped and the code to do it must be entered or updated in the svn repository (under OPS/PGExxx/byteswapping)
4. Are there any overall organizational differences between the old and new versions? (e.g. cross-granule where it was not, or vice versa)?
5. Most of the library modules under PRO are already imported into the NPP2 repository and have been altered to run in our environment. Ditto the include files associated to them. *Don't pull files from those modules into your PGE directory.*

WHILE YOU ARE CODING

1. Use the built-in capabilities of the IDPS code to read the ProCmnDictionary files, xml configuration files, and so forth needed by the module. PGE353 is a good example to look at. **NOTE:** You can set environment variables relating to which XML files and text files to read from within the C++ code; there is no need to do so from the perl script (See PGE353).
2. Note that the new libraries in NPP2, both B48 and Mx4, have the capability to look for alternate swath names, grid names, and field names. See these:
 - a. HDFGridFile::HDFGridVar *getVar(stringvec gridNames, string varName)
 - b. HDFGridFile::HDFGridVar *getVar(string gridName, stringvec varNames)
 - c. HDFGridFile::HDFGridVar *getVar(stringvec gridNames, stringvec varNames)
 - d. HDFFile::findDataField(stringvec fieldNames);
 - e. HDFData::findDataField(stringvec fieldNames)
 - f. HDFData (string path, stringvec swathNames, HDFmode rw, viirsDataType dataType = SDSType), HDFData::findSwath(stringvec swathNames)
3. **LUTs and LUT Structures**
 - a. Keep in mind that you need to build for both 32- and 64-bit machines; that means you may need to specify a preprocessor pragma pack(4) in some cases. When you do, you should surround it with preprocessor "ifdef LINUX" switches. See headers in the PRO/MIIRS/SDR libraries for some pragma examples. Keep in mind you need to do this before you byteswap.

- b. Do your LUTs have the correct version number on the end of their names? Please, no LUTs without versions.
- c. Have you used macros to specify any numbers (e.g. number of cross-granule rows, a dimension, etc.)? DID YOU CHECK TO SEE WHETHER THE QUANTITY ALREADY EXISTS AS A MACRO, either determined by the LPEATE or by IDPS?
- d. Have you byteswapped any new/changed LUTs or config files?

4. Configuration file environment variables and makefile flags

Note that the automatic setup in NPP2/Config, or NPP2/branches/Mx4/Config, triggered by dev_NPP_OPS_env_setup or dev_NPP_SCI_env_setup is subject to some command-line control but also automatically sets some environment variables.

- a. **Command-line control** includes using the “debug” and/or, for B48 the “MODIS_TILE_SIZE”, commands on the command line, which is what the specific setup files “dev_NPP_OPS_env_setup_debug”, “dev_NPP_OPS_env_setup_modtile”, “dev_NPP_OPS_env_setup_debug_modtile”, and “dev_NPP_SCI_env_setup_debug” do (there are no SCIENCE setup files specifying MODIS tile size because that is the only option for Science files). Also note that for dev_NPP_SCI_env_setup files only, additional special science flags may be used on the command line. See the header of the dev_buildlibs_SCIENCE in the NPP2/Library/buildfiles or NPP2/branches/Mx4/Library/buildfiles area for more details.
- b. **Automatic control** in the Config files sets environment variables appropriately depending on
 - i. Whether the machine being used is a 32-bit or a 64-bit machine (This works for Linux only, but the Land PEATE machines are all Linux machines)
 - ii. Whether the machine is a big- or little-endian machine (done by checking the node name; it is expected that those using a non-Land PEATE machine will edit the file appropriately)
 - iii. The file “Config/npp_buld_env_linux.csh” assumes the machine being used is a Linux machine and sets an environment variable to indicate so, the line for which is: “setenv LINUXNESS YES”. If the machine being worked on is not Linux, then this line should be edited to set LINUXNESS to “NO” instead.
- c. **Makefile flags** are set appropriately by the LP_SetMode.mk file located in Library/buildfiles which is included by Land_PEATE.mk. The makefile flags which are set are
 - i. The flag `__32_BIT__` indicating a 32-bit machine
 - ii. The flag `__LITTLE_ENDIAN__` indicating a little-endian machine
 - iii. The flag `LINUX` indicating a LINUX machine
 - iv. The flag `_MODIS_TILE_SIZE_` denoting MODIS (rather than VIIRS) tile size (the default for Mx4)

- v. A flag indicating HDF5 version: `_V167_HDF5_` indicates HDF5 V1.6.7 whereas the flag `_V180_HDF5_` indicates HDF5 Version 1.8.0.

WHEN YOU ARE BUILDING

1. **Makefile:** Does your makefile conform to the new NPP2 paradigm?
 - a. Have you used the included makefile “LandPEATE.mk”?
 - b. Have you followed the makefile template in Templates?
 - c. Have you fixed the makefile so that dependencies can be generated?
 - d. Does your makefile construct different executable versions depending on debug/production mode, modis/viirs tile size, and 32- or 64-bit executable?
2. **Buildfiles:** Have you constructed a master build file and auxiliary build files in the PGExxx directory using extant PGEs in the NPP2 repository as examples?
3. **Library versions used:** Have you carefully edited the PGExxx_EnvSetup.csh file in the COMB directory? In particular, if there are library groups that your code does not depend on, you should edit out references to them.
4. **Perl scripts:** Does your perl script also provide for different executable versions depending on debug/production mode, modis/viirs tile size, and 32- or 64-bit executable?
5. **Metadata:** Are values determined for all the metadata in the structure HDFGridMetaData (for gridded output) or the HDFSwathMetaData class (for Level1/Level2 data)? You should be specifying as many of these as possible in your PGExxx.h include file. Others, such as GRings, might be copied from other files. Production times are calculated by the class modules themselves.
6. **Big- and Little-Endian considerations:** these will probably not affect your code unless you are dealing with the IDPS OPS code that decodes the raw RDRs, but it may if you are reading other “bags-of-bytes” structures out of an HDF5 file. If you are affected, use the compile-time flags (see above regarding makefile flags) to notify the preprocessor to take big/little endian into consideration. Examples abound in the RDR/SDR/GEO code; look in the NPP2/PRO/SDR/VIIRS library code for examples.

32- vs. 64-bit considerations: See above under “LUTs and LUT structures” for one set of considerations. Also, be aware that in a very few places, the IDPS code directly sets addresses as being UInt64, usually if it is passing an address to a FORTRAN module. When this happens, you must conditionally define the address as UInt32 or UInt64 using the compile-time flags and appropriate preprocessor directives.

7. **Other nitty-gritty:**
 - a. You need not worry about mentions of `ProCmnLogger::getLogger` nor comment out mentions of `ProCmnAlgorithm`.
 - b. The overall IDPS OPS Version for use by the code is set by the (Land PEATE) code module “`Idps_Ops_Version.cpp`” in the directory `NPP2/IDPS/PRO/CMN/ProcessingIO/src`. *Edit the version there and nowhere else.*

- c. Surround mentions of any sort of mutex or mutex mounts with the preprocessor directive “#if defined(_REENTRANT) || defined(_THREAD_SAFE)” and “#endif”
8. **LongNames and Shortnames:** Be sure to double check the LongNames and the ESDTs for your PGE’s outputs against the latest Land PEATE System Description Document.
9. **LUNs and Param names:** Many of these changed in the NPP2 repository. Verify that the values you are using are in line with those defined in the NPP2/OPS/Include/LPEATE_Data/NPP_LP_Params.h. Note that if a needed LUN is not available and must be added that the limit is 32 characters. Also note that if you must make a change to this file, it will require a redelivery of the OPS libraries.
10. **Coarse Product Generating Shared Code:** The NPP_CRS shared code is to be run for all L2 outputs. If the PGE you are working on does not call the NPP_CRS shared code, it will need to. See PGE303, PGE330 or PGE311 perl scripts for examples on how the code is called. Similarly for L2G and L3 PGEs whose code is based on MODIS code -- we now have an NPP_CRS_L3 shared code that is to be run for these products.

WHEN YOU ARE TESTING

- Be sure your code runs and produces the same output on 32- and 64-bit machines (exception is code which runs only on the 64-bit machines due to large memory requirements)

BEFORE YOU SUBMIT YOUR CODE FOR BASELINING

1. Did you run “astyle” on all your C/C++ code and include files? (Or the “doastyle” script? It is in the directory NPP2/Utilities.)
2. Did you update the PGE history file?
3. Did you include a sample pcf?
4. Did you include a dependency list file (not a ciList file)
5. Did you include a ciList file?
6. Did you include a README file that explains how to build and run the code? Please do not recycle the incredibly out-of-date files that include mention of “MODIS STORE” and so on. These are unnecessary; you should be directing users to use the Land PEATE setup files which do all of this automatically. There are lots of good README file examples in the repository. If your README includes mention of out-of-date material, fix it. If your PGE has special production rules for executing within MODAPS, like using profiles, make note of it at the bottom of the README. See PGE353 and PGE302 README files for an example.
7. Does all of your code have appropriate headers that follow those in NPP2/Templates? Be sure that the contact information/points of contact for the Land PEATE are up to date.
8. Does any IDPS code included with yours need a header because you changed it in some way? Please put it there.
9. Did you build in production mode and make sure that your submission to CM is geared to production mode?

CHANGING A LIBRARY AND/OR ADDING TO A LIBRARY GROUP

All libraries in the NPP2 and NPP2/branches/Mx4 repository must be versioned.

After a library revision is checked in with CM, if you make **any** changes to a library, you should do several other things when you check that revision in to the repository:

1. If the library version file in Library/buildfiles still reflects the latest baselined version number, then update that file by upping the revision number. (It might be updated already if somebody else, or you, has already changed the library.)
2. Update the library history file in the Library/COMB directory. You need to edit both the top and bottom portions of the file.
3. Generate and edit a dependency list file (the DEPENDS parameter when you source a build file will do that). Edit it to remove directory references above "NPP2/", and check it in (it will have been created in the Library/COMB/DependencyLists directory.)
4. Check the README file in Library/COMB to make sure it does not need revision. (It probably doesn't.)
5. If you added another library to a group, or if you moved a library from one group to another, edit <Group>_Libraries.directoryList in the Library/COMB directory. (e.g. when an IDPS library was deleted, "IDPS_Libraries.directoryList", the file containing the directory listing was edited.)
6. **This is a critical step:** *Be sure that you rebuild all libraries and all code modules in the repository in both debug and production modes and that all builds are successful. Repeat the experiment on a 64-bit machine if you have been working on a 32-bit machine, and vice versa. You need to do this before your library group is submitted for baseline!*

THE SCIENCE LIBRARY

The "LPEATE_SPECIAL" library directories under the lib32, lib64, and Library directories contain versions of libraries built with a set of flags different from the standard set. At this time only the SCIENCE library group can be built with a special set of flags. There is a "README.txt" file in lib32, and another one in lib64, which explain the LPEATE_SPECIAL library. To quote the README in the lib32 directory:

```
NPP2/lib32/LPEATE_SPECIAL -- Libraries compiled with one or more
special flags set by LPEATE. Occurs rarely, and only for
SCIENCE libraries. These libraries
all end with the suffix "P.L.<version number >.a"
```

```
LPEATE_SPECIAL LIBRARIES WERE BUILT WITH THE SPECIAL FLAG
"USE_L2G_INCLUDES".
```

Note this does **not** mean that the SCIENCE library group cannot exist except in the LPEATE_SPECIAL directory. In fact, the regular production compiles of the SCIENCE Libraries are stored with the compiled OPS and IDPS libraries. If you need to build the Science libraries with a special flag (e.g. USE_L2G_INCLUDES) simply say:

```
source dev_NPP_SCI_env_setup USE_L2G_INCLUDES
```

which will return these messages:

```
BUILDING SCIENCE ENVIRONMENT IN PRODUCTION MODE.  
BUILDING SCIENCE ENVIRONMENT WITH TILE SIZE SET TO MODIS TILE SIZE  
BUILDING SCIENCE ENVIRONMENT WITH SPECIAL_CFLAGS SET TO -DUSE_L2G_INCLUDES
```

or

```
source dev_NPP_SCI_env_setup debug USE_L2G_INCLUDES
```

which returns the messages

```
BUILDING SCIENCE ENVIRONMENT IN DEBUG MODE.  
BUILDING SCIENCE ENVIRONMENT WITH TILE SIZE SET TO MODIS TILE SIZE  
BUILDING SCIENCE ENVIRONMENT WITH SPECIAL_CFLAGS SET TO -DUSE_L2G_INCLUDES
```

Other possible “special” flags are TEST_NOISY, ISINUS1, USDEF, TRACE_OUTPUT, and PARAM_ALLOW_SPACE.