SMAP L4\_C SW Delivery 3. Memo

Rev: 2012-12-12T18:33:00-07:00

Author: Joe Glassy, contractor (Lupine Logic Inc.)

Email: um.glassy@gmail.com

**Overview**

The SMAP Level 4 Carbon (L4C) algorithm represents an incremental improvement in implementing the L4C ATBD’s description of the Terrestrial Carbon Function (TCF) daily global model logic. From a developers viewpoint, one of main goals of this release is to present the L4C logic within a more unified software framework, emphasizing an improved spatial treatment, and illustrating a representative use of FPAR and Freeze/Thaw inputs. This D3 version produces daily outputs on a 2D spatial domain, at half-degree geographic resolution (360 lines by 720 samples), for a single model year. L4C outputs generated with this D3 release are in binary (little endian) format, with 2003 chosen as a representative model year. Output binary image files use ‘float32’ (IEEE single precision real) pixels unless otherwise noted, and are structured as ‘band sequential’ (BSQ) images with the ‘day’ unit serving as the “band”. For each of the (1..365) bands (or “day” layers) present in a given output file, each such band represents a populated global extent 2D image. Where a given pixel value for a science variable is masked (non-terrestrial, or missing), it is assigned a value of -999.0.

**Details**

1. Key Capabilities and features Added or Modified in D3 relative to D2 and earlier:
   * The D3 version of L4C adds a 2D functionality (in geographic projection, half-degree resolution) relative to the 1D point-model implementation of D2 and earlier. This is a transitory step on the way to modeling at the full spatial resolution of 1km internally, with outputs posted to a 9km EASE-GRID.
   * The D3 version adds more fidelity to the core L4C model logic as specified in the L4C ATBD, including carbon pool spinup, implementation of environmental constraints (EC, in temperature and moisture) as well as use of a (AMSRE) Freeze/Thaw (F/T or FT) input for EC; in production we will use the SMAP F/T.
   * A memory manager facility (implemented in l4c\_mem.c) wraps all high level function calls to “calloc” and “free” for the purpose of ensuring robust and consistent use of session memory resources by the application. The memObj facility is summarized in more detail in the accompanying L4C D3 Release Notes document.
   * A flexible exception and event notification facility is integrated within L4C message logging (e.g. via l4c\_LogMsg() in the D3 L4C code. Its purpose is to harmonize exception handling, and enable automated post-run analysis and classification of both routine metrics and exceptilons, as well as helping troubleshoot and correct any runtime defects or out-of-bounds conditions.
   * Detailed L4C D3 source code documentation is available in the ./srcdoc directory, containing cross reference and function, variable and datatype indices generared using the standard cxref (v1.6) source-code static analysis tool.
   * L4C code has been given a more consistent “look and feel” throughout.
   * The current L4C D3 code set itself consists of ca 5,738 LOC, across (5) header files and 10 source code files.
2. SW Location:

https://smap-sds-cm.jpl.nasa.gov/hg/level4Carbon/

The above location is presumed as the final location, once the hand off to JPL and GMAO/NCCS staff is complete.The SMAP L4C at D3. Once registered, the L4C D3 code will be managed in the Mercurial software version and configuration control system.

No JPL/Mercurial tag has been assigned yet, but we anticipate it will conform to the L4 SM tag of “*D00300*”.

1. Libraries and Executable Location
   * Libraries . The L4C D3 code, as producing binary outputs, uses standard Linux OS v2.6.x libraries (libc etc). If present, it uses the “libuuid” library and header files typically at “./usr/include/uuid/uuid.h”
   * Header files. The L4C D3 code uses the HDF4 (or HDF5) header files, but not the associated runtime libraries. The HDF headers are used to define self-documenting scalar datatypes such as “int16”, “float32” etc.

* Build environment:
  + the L4C D3 code requires a standard (POSIX compliant) “C” compiler capable of producing “c99” level code (e.g. must accept the directive “-std=c99”. The gcc compiler was used as the development platform, at gcc version 4.1.2.
  + Prior to building execute “source g5\_modules” in top-level “src” dir. This will link to GMAO Base Libraries and set up the build environment on NCCS/Discover (including: comp/intel-11.0.083; mpi/impi-3.2.2.006; lib/mkl-10.0.3.020; other/comp/gcc-4.5)
* Build & execution instruction
  + A traditional “make” Makefile is supplied, as a template to build the L4C code set, yielding a single executable named “l4c\_tcf.exe” at the conclusion of the build. A rudimentary ‘test’ target is provided, e.g. “make test”, which requires a site-specific runtime input parameters (RIP) text file containing the name-value-pairs that comprise all external runtime inputs, directives, and outputs.
  + No separate “make install” step is present in the L4C D3 Makefile, since not enough was known about the specific runtime environments at the JPL and GMAO .
* Other supporting Executables / Tools / Scripts:
  + L4c\_reset\_out.shl : a simple bash script is included for the purpose of resetting the output directory tree prior to a given run, when it is important not to intermix the outputs from a prior run with a new run.
* Other Dependencies:
  + The L4C D3 requires access to the HDF4 or HDF5 headers for defining size-specific scalar variables such as “int16”. As a convenience, a set of HDF header files is included in this delivery ( in tar archive hdf4\_headerfiles.tar.gz) in case the test system does not have HDF installed.
* Additional information:
  + A complete L4C D3 source code manifest is included in the Appendix.
  + The memory requirements for the L4C D3 executable runtime vary during an execution session, with a high-water-mark of approximately 7.5Gb of RAM required. Full runtime memory consumption information is available in the session log file, as produced via memObj’s built-in reporting mechanism.
  + Standard POSIX I/O calls (fopen(), fclose(), fread(), fwrite() ) are each wrapped by higher level functions (l4c\_openfile(), l4c\_closefile(), l4c\_fwrite(), and l4c\_fread()) that perform internal error-trapping.
  + The L4C D3 executable requires approximately 5 minutes of wall clock time to complete a run of the baseline test case, when run on a stand alone Intel Xeon/Nehalem equipped server. Several development and test platforms were used for L4C development, with the low end platform being a Intel i7 3 Ghz quad-core machine with 8Gb RAM and the higher end platform being a 16 core compute server with 48Gb RAM.
  + The L4C D3 code implements a high-level interface to session management (**l4c\_InitializeSession()**, **l4c\_CloseSession()**), and event logging and exception notification system, via the **l4c\_LogMsg**() function. This system employs the following protocol that when followed (as it is in the code), enables easy post-run analysis of all runtime metrics and log traffic.
    - The l4c\_LogMsg() function implements a “printf” style parser that accepts a text string identifying the caller context, along with a formatted printf() style format string. L4C D3 session log traffic is routed to a dated session log file such as the one delivered or the baseline test case (example: “l4c\_tcf\_2012-12-12.log”.
    - L4c\_LogMsg() formatted messages support a custom protocol, where the printf style string may start with “message-state-classifier” code ( introduced using a ‘@’ symbol), followed by one of the following codes that indicates the severity of the event: {‘A’=advisory only , ‘W’=warning only, ‘E’=error, but recoverable, ‘F’=Fatal error that is not recoverable. Here are several examples:
      * L4c\_LogMsg(“myfunction”,”@A Image has %d rows\n”,nRows);
      * L4c\_LogMsg(“myfunction”,”@W Condition (%s) is out of bounds! \n”,conditionString);
      * L4c\_LogMsg(“myfunction”,”@F Fatal error on (%s) invalid parameter %f\n”,conditionString,myParam);
    - Fatal error handling is tightly integrated with the memory manager, allowing graceful shutdowns in the event of a non-coverable error. For example, when a “@F” directive appears at the start of an error message handled by l4c\_LogMsg(), the system internally parses this directive and immediately begins an orderly shutdown, that includes calling the built-in memory manager’s garbage collector (in l4c\_mem.c), that releases all heap memory variables. This prevents the executable implementing large arrays fro leaving large blocks of orphaned memory.
  + The directory layout used to develop and test the L4C D3 model with the baseline test data set is shown later in this document. The output directories qualified by a PFTn qualifier are used to store the L4C outputs as masked by the indicated Plant Function Type (PFT) classifier, mimicking the organization of production outputs. The input directories are {FT, FPAR, MERRA, PFT, SOC, and VI}, although at this time the VI data is not used.
  + The L4C D3 executable requires approximately 5 minutes of wall clock time on a stand alone Intel Nehalem equipped server with 48G. The Test Base produces approximately 20Gb of output data.

1. Test Cases and Datasets
   * Baseline Test Case summary
     + The baseline L4C test case consists of a global half-degree input dataset, derived from MODIS and MERRA data, covering a daily timestep execution period of one year. Data from a 2003 sample year was used.
   * Baseline L4c ancillary data elements:
     + Biome Properties lookup (LUT) table. The BPLUT LUT is a text file containing biome properties (runtime parameters), organized by Plant Function Type (PFT) as the sole lookup key. Note that the Plant Function Type classifier image was derived from the MCD12Q1.Land\_Cover\_Type\_5 (HDF4) file, converted to a binary file containing jus the dominant PFT class per cell at half degree.
   * Baseline L4C Input Data components:
     + The L4C input data elements are documented in the runtime input parameters text file accompanying this delivery
     + These were manually generated from higher resolution datasets down to half-degree resolution (720 columns by 360 rows).
   * Baseline L4C Output Data elements

Test Case

o Purpose of the test: Demonstrate a run of a single year at daily timestep, at half-degree resolution, using inputs derived from MODIS and MERRA.

o Path name to input files: note that all paths for input and output files are contained in the user configured runtime input parameter (RIP) text file of name-value-pairs. The RIP file syntax allows for comment lines introduced using a standard ‘#’ as the first non-whitespace character. The location of the baseline test case input and output files for the L4C D3 delivery is TBD, but is likely to mirror the placement of inputs for L4 SM. It will something like:

smap-tb.jpl.nasa.gov: /simulation/level4/v1/L4\_C/SMAP\_L4\_C\_D00300/test\_case/

o Run directory:

“./baseline”

execute “l4c\_tcf.exe <l4c.rip> ” at the command line

o Inputs and Outputs directory layout for L4C D3 baseline test case is:

./baseline/ancillary

./baseline/FT

./baseline/FPAR

./baseline/MERRA

./baseline/outputPFT0

./baseline/outputPFT1

./baseline/outputPFT2

./baseline/outputPFT3

./baseline/outputPFT4

./baseline/outputPFT5

./baseline/outputPFT6

./baseline/outputPFT7

./baseline/outputPFT8

./baseline/output

./baseline/PFT

./baseline/SOC

./baseline/VI

1. Change Requests and Problem Reports Addressed (n/a)

Appendix

|  |  |  |
| --- | --- | --- |
| **Level 4 Carbon (L4C) Delivery 3 Source archive Manifest** | | |
| Module file | Type | Comments |
| l4c.rip | Runtime configuration parameters | A commented D3 test case name-value-pair text file containing all input directives, and paths to input and output files |
| l4c\_tcf\_2012-12-12.log | Session log file example |  |
| Makefile | Build tool | Standard makefile, will need modification to run on NCCS or JPL systems. |
| l4c\_adt.h | Header,for abstract data types | Defines L4C abstract data types, enums, macros |
| l4c\_prot.h | Header, for L4C function prototypes | Defines L4C function prototypes |
| l4c\_nvp.h | Header, for NVP adt’s. | Defines L4C name-value-pair facility abstract data types, macros, enums. |
| l4c\_nvp\_prot.h | Header, for NVP function prototypes | Defines L4C name-value-pair facility function prototypes |
| l4c\_tcf\_halfdeg.c | Source, for “main()” | Defines L4C “main” and core spatio-temporal model logic loops |
| l4c\_tcf\_logic.c | Source, for all L4C TCF science logic | Defines L4C science logic for all carbon modeling functions and I/O functions. |
| l4c\_meta.c | Source, for metadata handling | Defines L4C metadata handling logic. This is just a stub for D3. |
| l4c\_bplut.c | Source, for BPLUT management | Defines Biome properties lookup table (BPLUT) read I/O, lookup access and mgmt. functions. |
| l4c\_qa.c | Source, for QA implementation | Defines QA / QC handling code. This is only a stub at D3. |
| l4c\_log\_svc.c | Source, for log message and event handling | Defines log messaging services, event traffic handling, integrated with memObj() on fatal errors. |
| l4c\_rip\_str.c | Source, for runtime input parameter (RIP) name value pair management | Defines runtime input parameter name-value-pair parser and handling code for L4C D3. |
| l4c\_mem.c | Source, for the memObj() facility | Defines memory manager logic with garbage collector; e.g. all memObj() logic: memObjCalloc(), memObjFree(), memObjGarbageCollector(), and related functions |
| l4c\_util.c | Source, for L4C “mini API” session management and utility functions | Defines L4C D3 session management and utility functions. |
| l4c\_etimer.c | Source, for event timer code. | Defines L4C D3 internal event timer logic. |