Quick guide to upgrading from CGNS v2.5 to v3.x

This sheet is intended as an aid for people who already use CGNS v2.5 or earlier, who are not sure what is involved in upgrading to v3.x. The major new capabilities in CGNS are 64-bit integers (starting in v3.1) and parallel I/O (starting in v3.2). 64-bit integers may be necessary for use with very large grids (hundreds of millions of nodes or larger) for which cell-to-node pointer lengths (for example) may exceed the 32-bit integer limit. This sheet also addresses in note 2 how to use v3.x to write CGNS files that are readable by older (v2.5) CGNS software, if desired.

1. If you do not need 64-bit integers, then build CGNS v3.x using default 32-bit (if your compiler complains about cgsize_t in the function prototypes, then use the "enable-legacy" compile option which defines cgsize_t to be an int).

2. If you want to be able to write new CGNS files that can be read using older software that only supports v2.5, then you must build CGNS v3.x using 32-bit (because v2.5 and earlier did not support 64-bit integers). Furthermore, in your code, use the call cg_set_file_type to set the CGNS file type to CG_FILE_ADF2, or set the environment variable CGNS_FILETYPE to be CG_FILE_ADF2 prior to execution. This will force v2.5 compatibility. The call cg_set_file_type is a run-time command issued before opening the CGNS file with cg_open. See: http://cgns.github.io/CGNS_docs_current/midlevel/fileops.html. Note that CGNS software can always read older CGNS files (backward compatibility is ensured), but CGNS files written using a major new release are typically not readable by older CGNS software (forward compatibility is not ensured).

3. If you need 64-bit integers (because your grids are very large or because your code uses 64-bit integers), then build CGNS v3.x using "enable-64bit" (and without "enable-legacy"). Your software that interacts with CGNS must also use 64-bit integers as appropriate. Resulting CGNS files will not be readable by older CGNS software.

4. If you previously used cg_section_partial_write (with 10 arguments in C, 11 arguments in Fortran), note that a change was made starting in v3.1 to use cg_section_partial_write (with 9 arguments in C, 10 arguments in Fortran) plus cg_elements_partial_write_f. Be sure to account for this non-backward-compatible change! See: http://cgns.github.io/CGNS_docs_current/midlevel/grid.html#elements.

5. If you have possible conflicts between items in the CGNS header file (cgnslib.h) and other header files (both headers define "Vertex" for example), then build v3.x using "enable-scope." This enables scoping of all enumeration values in cgnslib.h by prefixing them with a CG_. If this is used, then your code will need to access the enums via (e.g.) CG_Vertex, but "Vertex" will still be written to the CGNS file itself. Thus, scoping affects the user code, but does not affect the CGNS files themselves.

6. To use parallel read/write, you must compile CGNS and HDF5 using MPI, and build CGNS with option --*enable-parallel* and build HDF5 with option --*enable-parallel.*

7. Discussion of 64-bit portability issues is provided on: http://cgns.github.io/CGNS_docs_current/midlevel/general.html#c-port. Also, change files (changes_from_2.5.txt and changes_from_3.0.html) are provided with the source code distribution.

8. Note that as of V3.3, support was dropped for non-Fortran 2003 compliant compilers. There have also been some other Fortran-related updates. For example, "include cgnslib_f.h" has been changed in favor of use of a module "use CGNS". To make use of the newer features of CGNS, this should be adopted. See, for example, Fortran files named *_f03.F in the src/tests directory. However, CGNS is backward compatible, and the old "include cgnslib_f.h" is still allowed.