This document contains information from NCAR Software Engineers working on the SingleTrack project regarding requirements for the NCAR Physics Driver Framework. These requirements were built on the CCPP requirements, following this color code:

- Black: requirements previously formally accepted by the NGGPS Program Office.
- Green: Rewording of requirements previously formally accepted by the NGGPS Program Office
- Blue: NCAR-contributed requirements that were implicit in the initial design of the CCPP framework and CCPP v1 release.
- Magenta: NCAR-contributed requirements that were not planned for in the initial requirement set of the CCPP. These requirements were reviewed by GMTB staff and considered to represent useful enhancements to the UFS in support of R2O.

Last modified May 2018

ID	Item	Reason
D1	The driver shall allow parameterizations to be agnostic of host application.	Well-established convention facilitates data mapping.
D2	The driver shall provide an easily configurable entry point for passing information to/from physics parameterizations.	Enhances portability and simplifies the interface for community contributions.
D3	The driver shall be expandable to include new variables.	Newly added parameterizations may need information not already provided by host application.
D4	The driver shall provide the ability to select different parameterizations of the same category via an external option selection.	Provides flexibility and ease-of-use; allows direct comparison between schemes, possibly within an existing suite.
D5	The driver shall allow parameterizations to be used as suites or be selected individually.	Suites are useful in both an operational and research environment; the ability to choose individual schemes is important for testing and development.
D6	The driver shall allow the order and frequency of calls to individual parameterizations to be configurable.	Allows for sensitivity testing of different physics configurations.
D7	The driver shall provide the capability to share the same instance of physical constants with all model	Maintains consistency among model components.

	components (host application and parameterizations).	
D8	The driver shall include documentation including references, functional descriptions of code, guidance for how to call parameterizations as suites or individually in any order, and guidance on how to connect new parameterizations or host applications.	Community code should be well-documented for users and developers.
D9	The driver shall be developed using modern and robust coding standards balancing portability, computational performance, usability, maintainability, and flexibility, and follow coding guidelines listed here.	Following Kalnay rules (Kalnay et al., 1989, BAMS and Doyle, J. D, M. Iredell, P. Tripp, J. Dudhia, T. Henderson, J. Michalakes, J. A. Ridout, J. Rosinski, S. Rugg, R. Adams Selin, T. R. Whitcomb, K. Lutz, and D. McCarren, 2015. Revisiting Kalnay rules for physics interoperability: 25 years later. AMS Eugenia Kalnay Simposium, Jan 5-8, Phoenix, AZ)
D10	The driver shall allow the ability to drive parameterizations or suites in "offline mode".	Offline mode allows for sensitivity and process-based studies; removes response from other components to focus on impact from parameterizations.
D11	The driver shall provide the ability to pass arbitrary collections of column variables to parameterizations.	Follows modified Kalnay rules 6, 7. Increases computational performance.
D12	The driver shall provide the ability to deliver variables computed by, or for, use within any parameterization for diagnostic purposes to the model I/O component.	Important for testing, development, and evaluation. Note that the delivery to model I/O component will be done at the host application level.
D13	The driver shall provide the ability to deliver variables computed by, or for, use within any parameterization to external models.	Facilitates consistency with other Earth System models (e.g. ability to share roughness length between parameterizations and LSM). Note that coupling to external models will be done at the host application level.
D14	The driver shall not modify answers produced by the parameterizations.	Eliminates inadvertent errors.

D15	The driver shall allow run-time specification of parameters (possibly greater than 1D).	Allows rapid tuning and sensitivity experimentation.
D16	The driver shall ensure that more than one scheme variable with the same physical meaning but different names cannot exist.	Minimizes ambiguity.
D17	The driver code management and support shall be designed so community scientists can use and propose contributions.	Meets the NCEP goals for community modeling and enhances R2O.
D18	Parameterizations can specify to the driver what fields it requires, which it generates (or modifies), and which fields it 'owns' (if any).	Some parameterizations provide data for other parameterizations. but require that other parameterizations do not modify the field.(This is analogous to the Fortran protected idea).
D19	Parameterizations and physics driver must be able to communicate information to allow the host model to write state to restart, restart from previously written files, and to reset the parameterization internal state during a run from restart files.	The restart requirement is essential both to long runs and to CESM data assimilation.
D20	The physics driver must be able to be multiply-instantiated (i.e., one run supports multiple, independent physics suites).	This allows a model to call, for example, a chemistry package on a different time scale and possibly different grid than the mainline physics suite.
D21	The physics driver must be able to receive error codes from any parameterization and pass them to the host model for output and model termination.	If we are discouraging parameterizations from doing I/O and from stopping the model, we must pass this information to the host model
D22	The physics driver must enable scientists and performance engineers to inspect the code that is part of the model run.	Provides traceability.
D23	The physics driver must support the dynamic generation of tracing information	Scientists and developers need a way to see procedure calling sequence and data flow.
D24	The physics driver must be able to	Many chemistry packages operate on arrays of

	communicate arrays of variables (e.g., an array of tracers) whose extent is only known at compile time.	species which may only be known at compile time. The metadata for each species must be passed through the driver from this host model to the parameterizations.
D25	The physics driver system must be able to handle fields with multiple time levels.	For example the dynamics may have two time-levels, and the physics may have before and after values of a variable.
D26	The physics driver system must be able to collect data at a specified point (e.g., before a target parameterization) and pass this data to the host model for output.	Offline studies are extremely useful for parameterization development and tuning.
D27	Run-time selection of physics suites and schemes	Regression testing, multi-physics ensembles
D28	The physics driver system must be able to operate both on systems that support and do not support dynamic library loading.	We need to allow for either capability to ensure that the framework can be run on systems to that do not support dynamic libraries.
D29	The driver must support passing of derived types between host model and physics schemes	When porting new physics that originally used DDTs in their argument list, it may be helpful to call the schemes exactly as they were called. Also, schemes may call libraries that store, e.g., state in a derived type.
D30	Driver supports multiple kind-types for integers and reals	Mixed-precision physics
D31	Driver supports calling of generic physical parameterizations (including land-surface, simple ocean, combined schemes, purely diagnostic schemes)	Capability to call land-surface model is required for stand-alone WRF and MPAS. Some schemes are not easily binned into a traditional physical parameterization scheme taxonomy.
D32	Host model and driver may interrogate each other during a “handshaking” phase	At run-time, the host model may need information about physics choices in order to properly allocate memory; the driver may need information on which fields are available in the host model
D33	The driver shall provide “hooks” to facilitate debugging and general scheme development	For example: A developer can call a developer-defined function before and after each call to a physics scheme to enable checking of field values.
D34	Recommendation: Internally, the	This would facilitate, e.g., range checks on fields

	driver should support the association of arbitrary (and extensible) metadata with fields	before and after calls to schemes (through hooks), tracking of data flow in the driver, and it may be used to implement matching of units, long_name, etc. of fields
D35	The driver system must support processing (e.g., averaging, max/min selection) of diagnostic fields which may be output multiple times during a run.	Parameterizations are often called multiple times during a suite time step. Since the host model is not active during this time, the driver (and / or parameterizations) must be able to handle multiple diagnostic outputs.

CCPP physical parameterization requirements

The primary purpose for each of the physical parameterizations within a model is to advance the solution of specified tendencies or state variable. The CCPP relies on a separation of concerns between physics and dynamics, and it is recommended that parameterizations do not deviate from this primary purpose.

Note is that the operational FY17 GFS suite has been grandfathered and is not required to meet these requirements.

ID	Item	Reason
C1	Physics schemes shall conform to standard variables.	Common variables facilitate scheme portability. We eventually need a list of acceptable variables.
C2	The CCPP shall allow multiple parameterizations that represent a physical process or processes to coexist in the CCPP.	The CCPP can support all NCEP needs (including research and development).
C3	Transparent criteria shall be used to guide number and choice of parameterizations included in CCPP.	A Change Review Board reviews test results and ensures quality control of parameterizations and has authority over portfolio of supported parameterizations. Maintenance is kept to a manageable level while focusing on operational and research applications.
C4a	The CCPP schemes shall have standard and documented testing procedures and metrics applied by all	The Change Review Board defines minimum testing procedures and metrics. This may include specific codes/tools to be employed in the test

	physics developers.	harness.
C4b	CCPP compliancy is verified with provided tools. Guidance is provided concerning how to add new parameterizations.	
C5	The CCPP schemes shall have standard and documented observation and model databases for testing.	Both observation and model-generated datasets need to be selected and available for testing. This ensures that the Change Review Board has material that is easy to judge. Tools to subset or process data may be part of this, as necessary.
C6	The CCPP schemes shall permit parameterizations to expose all tunable parameters	Tunable aspects of parameterizations will be configurable by run-time settings, e.g. Fortran namelists, allowing a single software instance of a parameterization to satisfy all foreseeable models and applications.
C7	The CCPP schemes shall permit a capability to share same instance of physical constants with all model components (host application and parameterizations).	Maintains consistency among model components.
C8	The CCPP schemes code management shall be designed so community scientists can use and propose contributions.	Meets the NCEP goals for community modeling and enhances R2O.
C9	The CCPP schemes shall have documentation including references, functional descriptions of code, information on inputs/outputs to parameterizations.	Community code should be well-documented for users and developers.
C10	The CCPP schemes shall employ modern and robust coding standards supporting portability, computational performance, usability, maintainability, and flexibility and follow coding guidelines listed in the Coding Standards.	Follows modified Kalnay rules.
C11	At initial, restart, or finalize time parameterizations may read non-decomposed data (such as look-up tables) that do not have scope outside	Schemes should only perform I/O during identified phases. It is preferable that I/O is done in a scalable way whenever possible (e.g., read and broadcast from one process, rather than have all

	of the physics scheme.	processes read the same file). If MPI routines are used for parallel I/O or for broadcasting information, the code must be guarded in CPP directives and provide alternative ways for systems/models where MPI is not used.
C12	Parameterizations must pass a return code to the driver to indicate success or failure.	This allows status communication without the parameterization trying to write log messages.
C13	The system that translates the parameterization metadata cap into a driver-callable interface must produce Fortran code as part of the preprocess step.	Scientists and performance engineers must be able to inspect the code that is part of the model run.
C14	A physics parameterization must be able to conditionally compute a diagnostic field depending on whether or not the field will be output.	Some diagnostic calculations are expensive.
C15	The number of OpenMP threads a scheme is allowed to use internally must be provided as an input argument to the scheme.	The host application cap (or the upstream calling application) can have OpenMP parallelism and/or allow physics schemes to use OpenMP internal regions.
C16	The physics schemes shall only use MPI for broadcasting non-decomposed information (such as from a look-up table) to the rest of the group on the communicator. Furthermore, this functionality is restricted to initialization.	Currently, 3d physics will be enabled by chopping scheme in multiple pieces, where the host cap is responsible for traditional interprocessor communications.
C17	Every physics scheme entry point (interface routine) is documented with specially formatted metadata.	The metadata is used to build “cap” routines to be called by the framework.
C18	No message passing communication between columns is allowed during a parameterization’s run-time execution. Read-only access is acceptable from stencil-defined neighbors.	Physical parameterization schemes that assume 3d capabilities will be supported via breaking up the scheme into multiple parts. The first part of the scheme is handled, the data returns back up to the host application, communication is performed, and the data is sent back down into scheme part #2. As usual, the host application cap and the shared driver are involved in the data transfer.
C19	Schemes must follow standardized naming for entry points.	Calls to schemes may be automatically constructed, and we’d need to be able to identify,

		e.g., the init, compute, and finalize entry points
C20	Parameterizations can specify to the driver what fields it requires, which it generates (or modifies), and which fields it 'owns' (if any).	"Owns" refers to parameterizations that provide data for other parameterizations, but require that other parameterizations do not modify the field. "Requires" refers to fields that must be provided as input to the parameterization "Generates" refers to fields that must be provided as output to the calling driver.
C21	Each scheme must provide vertical index ordering to the driver.	Models have different vertical index orderings as do physics parameterizations. Parameterizations must know the vertical ordering of its interface fields so that they can convert if they use a different vertical ordering.
C22	Each scheme must communicate index ordering (which index is horizontal dimension, which index is vertical dimension, etc.) to parameterizations.	Parameterizations must know the ordering of input and output data so that it can do data rearrangement if necessary.
C23	A parameterization needs the ability to specify to the host model which species will be transported.	A chemistry package may have species which are not to be advected.
C24	Parameterization metadata should allow a field specifying allowed field ranges.	As an example, a temperature field could generate an error if outside the range of 100K -- 345K. Could be used with D34.
C25	Metadata for certain fields (e.g., pressure, constituent mixing ratios) must indicate whether they are wet or dry and, if wet, which water phases are included.	Physics parameterizations need to either make calculations based on the nature of the pressure field or to convert the pressure field (e.g., to dry) before doing calculations. This requirement applies to any field whose numerical value depends on the use of water vapor and condensed water species.

Host application cap requirements

Note that the host application cap (FV3GFS cap) used in the first release of CCPP already meets the requirement below

ID	Item	Reason

H1	The host cap must be able to interface directly with both the host application and the physics driver. The host model cap must be able to register variables in the driver's data structure (including the location of the variable in memory and other metadata).	The physics driver assumes various C to Fortran constructs, and assumes Fortran module constructs. Currently, the host applications that are targeted are written in Fortran. Note that this replaces the previous H1, as it is is an extension of previous NGGPS H1 requirement for additional flexibility.
H2	The host application cap (or the upstream calling application) shall manage all variables that are arguments to individual parameterizations. Manage includes, but is not limited to, allocation, distributed communication, initialization, I/O, correct numbers of scalars, and metadata.	For consistency, it is the task of the upstream systems to manage the variables in the subroutine argument lists.
H3	The host application cap (or the upstream calling application) shall perform all required I/O, except for a few notable cases (see C 11).	The fundamental purpose of a physical parameterizations is to compute some specific physical process. The more the parameterizations stay aligned with this standard, the more portable the schemes are. Allowing complicated I/O systems to be introduced into parameterizations reduces the chance at simple portability of those schemes with a new Host Application. The original requirement was trimmed for clarity.
H4	The host application cap (or the upstream calling application) shall handle all required run-time distributed memory processing for decomposed arrays, if any is required. Upon entry into each parameterization, each input field in the argument list is assumed to be the correct value to use.	The physical parameterizations do not carry information that allows them to determine neighboring grid columns. The parameterizations are all assumed to be 1d columns of independent data, though for performance purposes, blocks of those 1d columns may be bundled into arrays. The original requirement was trimmed for clarity.
H5	The host application cap (or the upstream calling application) shall handle all processing that requires that the	For ease of portability, the parameterizations only know the arrays provided in the argument lists, the provided array sizes, and the computational extent for each of the arrays. The parameterizations are not

	parameterizations be computed on a grid or resolution different than the upstream calling application.	aware if the incoming arrays are indeed the original size of the grid that the Host Application is running.
H6	The host application cap (or the upstream calling application) shall handle all processing that requires that the parameterizations be run concurrently.	The physical parameterizations have no information about the sequential nature of their own processing, other than the list of arguments defined as either input or output. Because all information for a parameterization comes through the argument list, the parameterization is well suited to being insulated from external processing techniques. The upstream calling application has the necessary software infrastructure tools to set up concurrent parameterization processing.
H7	The host application cap (or the upstream calling application) can have OpenMP parallelism and/or allow physics schemes to use OpenMP internally. The number of OpenMP threads a scheme is allowed to use internally must be provided as an input argument to the scheme.	The cap for the physical parameterizations is automatically manufactured. For timing performance and portability, all OpenMP threading for a particular scheme is controlled by each scheme's cap.
H8	The host application cap shall use Fortran array syntax that is valid for arguments that have an explicit interface.	Taking advantage of argument mismatch for type, kind, and rank is only available with explicit interfaces. Given that the purpose of the effort is to include additional schemes, allowing the compilers to find argument mismatches is a benefit.
H9	Host cap metadata shall use the same standard names as used by parameterization metadata.	Common variables facilitate scheme portability.
H10	Metadata for certain fields (e.g., pressure, constituent mixing ratios) must indicate whether they are wet or dry and, if wet, which water phases are included.	Physics parameterizations need to either make calculations based on the nature of the pressure field or to convert the pressure field (e.g., to dry) before doing calculations. This requirement applies to any field whose numerical value depends on the use of water vapor and condensed water species.
H11	Host must communicate its internal vertical level and indexing (e.g., index 1 equals	Models have different vertical index ordering as do physics parameterizations. Parameterizations must know the vertical ordering of its interface fields so that

	model top). It must either use the model vertical level or convert to a 'standard' vertical level. In either case, this level information must be communicated to the parameterizations in case they use a different vertical scheme.	they can convert if they use a different vertical ordering.
H12	Host must communicate index ordering (which index is horizontal dimension, which index is vertical dimension, etc.) to parameterizations.	Parameterizations must know the ordering of input and output data so that it can do data rearrangement if necessary.
H13	The host model system must support user access to every parameterization's exposed tunable parameters.	For example, if a parameterization reads a namelist, the host model must pass it

Suite functionalities and the Suite definition file (SDF)

ID	Item	Reason
S1	The SDF is human-readable, and also easily parsed by a computer.	
S2	The SDF identifies the names of the schemes to call, and the order in which the schemes are called.	
S3	The SDF allows multiple levels of grouping and partitioning of schemes.	Physics called in multiple places within the host model, sub-cycling, etc.
S4	The SDF shall differentiate between a scheme's phases such as "init", "run", and "finalize".	

S5	The physics suite definition must have explicit support to define both process split and time split sequences as well as shadow parameterizations.	• In a process-split sequence, two or more parameterizations are called on the same initial state and their output tendencies are added to produce the updated state. • In a time-split process, each parameterization is called with a state updated by the previous parameterization. • Shadow parameterizations sample the model state but do not update the state or contribute to tendencies.

Coding standards

STATUS: The proposed standards here need to be reconciled with Environmental Equivalence 2. The section should be updated to reflect the standards that NCEP/EMC intends to follow.

The following table specifies coding requirements and recommendations for a parameterization to be included in CCPP. The intent is to promote readability, robustness, and portability without being too onerous. The Kalnay rules and work by the NUOPC Physics Interoperability Team and EMC personnel had a major impact in creating this list. The GSM coding standards described at https://svnemc.ncep.noaa.gov/trac/gsm/wiki/GSM%20code%20standards were taken into account and incorporated as applicable. Unless specified otherwise, the Fortran programming language is assumed.

ID	Type	Item	Reason	Source	Status
CS1	Required	All modules or subroutines will contain “implicit none”	Assists in writing bug-free code. Understanding implicit type rules is difficult and arcane. Understanding where a variable comes from (local, input argument list, module) is more difficult with implicit typing	GMTB, GSM	
CS2	Required	All arguments to subprograms will contain the “intent” attribute. All intent(out) variables must be set in the subprograms.	Assists readers in understanding whether a variable is read-only (intent(in)), read/write (intent(inout)), or effectively uninitialized (intent(out)). A compiler error will result if code attempts to use a variable differently than specified in its “intent”. Declared variables without the “intent” attribute can be understood to be local.	GMTB, NUOPC PI Team, GSM	

			Not initializing intent(out) variables can lead to different answers, depending on the system/compiler/optimization level, and make debugging difficult.		
CS3	Required	No modules or subroutines will violate the Fortran 2008 standard	Makes porting to a new compiler easier to near trivial. Example: gfortran by default enforces the standard that free-form source lines will not exceed 132 characters. Some compilers by default allow line lengths to exceed this value. Attempts to port codes with line lengths greater than 132 may encounter difficulty.	GMTB	
CS4	Required	All local and argument list variables will have a comment explaining the meaning of the variable. An in-line comment on the declaration line is sufficient	Allows readers unfamiliar with the code to more quickly understand how the code works.	GMTB, NUOPC PI Team, GSM	
CS5	Required	All modules and subprograms will have a documentation block describing functionality	Promotes understanding of algorithms and code structure by new users	GMTB, NUOPC PI Team, GSM	
CS6	Required	Common blocks are disallowed	Deprecated Fortran feature. Modules provide all the functionality of common blocks plus much more.	GMTB, NUOPC PI Team, GSM	
CS7	Required	A package must be compilable with the gfortran compiler (or gcc for packages coded in C). Runnability and validation can be provided using whatever compiler(s) the developer prefers.	gfortran (and gcc) is free and ubiquitous, and therefore is an ideal choice for canonical compiler.	GMTB	
CS8	Required	All Fortran source will be free-form	Fixed-form source is hard to read and archaic. A 72-column requirement only makes sense for punch cards.	GMTB	
CS9	Required	All public subprograms will be Fortran-callable	Fortran is the most commonly used language for geophysical models.	GMTB	
CS10	Required	All parameterizations must be thread-safe (except for initialization and finalization methods)	Many geophysical numerical models are threaded these days, and need to be able to invoke physical parameterizations simultaneously from multiple threads. Example code which is NOT thread-safe: Declare a variable “first” and initialize it to .true. Then test its value and set some static variables if it is .true. This will likely result in wrong answers when run in threaded mode. Solution: Provide an initialization routine which sets the static variables outside of threaded regions. Wikipedia provides a brief overview of	GMTB, NUOPC PI Team, GSM	

			thread-safety: https://en.wikipedia.org/wiki/Thread_safety		
CS11	Required	No parameterization will contain a “stop” or “abort” clause	If an error condition arises, it is better to set a flag and let the caller decide how to handle the condition.	GMTB, NUOPC PI Team, GSM	
CS12	Required	Use of uninitialized variables is disallowed	Readability. Not all compilers can be made to initialize static or stack variables to a known value (e.g. zero).	GMTB, GSM	
CS13	Required	All array indices must fall within their declared bounds.	Debuggers will fail when “tricks” are employed which reference arrays outside of their declared bounds.	GMTB, NUOPC PI Team, GSM	
CS14	Required	Multiple runs of the same compiled parameterization given identical input must produce identical output. In the case where randomness is part of the parameterization, a method must be provided to invoke the same random sequence for test reproducibility.	Prevents inadvertent errors.	GMTB, GSM	
CS15	Required	The use of compiler flags specifying default precision is disallowed. For example, if 64-bit precision is required, use the “kind=” attribute to specify the precision rather than a compiler flag such as “-r8”	The behavior of flags is compiler-specific, e.g. if the user specifies real*4 does the -r8 compiler flag mean the variable is real*4 or real*8?	GMTB	
CS16	Recommended	With the exception of common libraries which use a well-defined naming standard for variables and subroutines, all module “use” statements must explicitly state which public entities will be referenced. The MPI library is an example of an acceptable exception: All MPI routines start with “MPI”, so a blank “use mpi” statement is acceptable.	Assists in understanding where various variables and/or functions or subroutines are defined.	GMTB	
CS17	Recommended	All code intended for debugging purposes only should be removed prior to submission for inclusion	Readability	GMTB, GSM	
CS18	Required	All arrays explicitly allocated, must be deallocated when no longer needed	Readability. Minimize memory usage.	GMTB, GSM	
CS19	Reco	The default visibility rule for	Limiting variable and subprogram scope is	GMTB	

	mmen ded	module variables and procedures should be “private” (specified by a single “private” statement near the beginning of the module). The “public” attribute is applied to only those entities which are needed by other subprograms or modules.	good programming practice		
CS20	Reco mmen ded	Consistent use of case is preferred for Fortran code (text strings excepted).	While Fortran is a case-insensitive language, variable “aBc” should also be expressed that way, and not “aBc” in one place, “abc” in another, and “ABC” in another.	GMTB	
CS21	Reco mmen ded	A parameterization should contain “init”, “run”, and “finalize” methods. The “run” method must be thread-safe.	Promotes separation of activities which must be done only once at startup or shutdown, from those which are done on multiple time steps.	GMTB	
CS22	Reco mmen ded	Parameterizations should be able to be invoked in “chunks”, where the calculations are independent of the fastest-varying subscript.	Computational performance is the main reason for this preference. Many physical parameterizations in geophysical models contain a dependence in the vertical, which means this dimension is unavailable for vectorization. Vectorization can provide up to a 16X speedup on modern processors. Example: Outer loop over vertical index “k” can contain vertical dependence, but if there is also an inner loop over horizontal index “i” that can be vectorized, the code is likely to run much more efficiently.	GMTB	
CS23	Reco mmen ded	The use of “goto” is strongly discouraged, except where no better option is available.	Modern languages provide better mechanisms to accomplish the same goal in most cases.. “goto” promotes “spaghetti” code, which can be unreadable.	GMTB	
CS24	Reco mmen ded	Code and declarations within subprograms, loops, and conditional tests should be indented. Indenting by 2 or 3 or 4 columns is reasonable. Spaces are to be used for intents, tabs are disallowed.	Readability. Particularly important for multiply nested loops and/or “if” tests.	GMTB	
CS25	Reco mmen ded	Test operators <, <=, >, >=, ==, /= are preferred vs. their deprecated counterparts .lt., .le., .gt., .ge., .eq., .ne.	The modern constructs are easier to read, and more understandable for those unfamiliar with legacy code.	GMTB, GSM	
CS26	Reco mmen ded	The use of bare constants (e.g. 2.7) inside of computational regions is strongly discouraged. Instead, a named constant (e.g. some_variable = 2.7) should be declared at the top of the routine or module, along with an in-line comment stating its purpose	Bare constants buried in code is one of the biggest contributors to lack of readability and understanding of how code works. “What the heck does 2.7 mean???” In addition, using a named constant makes it easier to specify precision, e.g. real*8 some_var = 35.	GMTB	