1 What's in this file

Only changes to the ECHSE's 'core sources' are listed in this file. You won't find information about changes made to a user-defined class (i.e. a specific model).

2 Changes to the code

2014-05-07: Multi thread control

The name and meaning of the config keyword to control the number of threads has been modified. Instead of a logical value after keyword `multithread` one has to supply the desired number of threads now after keyword `number_of_threads`. Values less that 1 will be changed into 1. If the requested number exceeds the maximum possible number of threads on the particular machine, the value is reduced to the maximum possible number.

2014-02-27: Multi-run mode no longer supported

The multi-run mode (see log entry from 2012-01-12) is no longer supported and the source code was cleaned accordingly. In tests, the multi-run mode did not show the expected gain in performance (possibly due to slow system calls). In addition, the interaction between the model and the necessary external programs/scripts turned out to be overly complicated in practice. Finally, the multi-run mode was not useful in the context of optimization based on external tools. In those cases, the ECHSE-based model runs inside the objective function as a black box and it is expected to produce a single result from a single input data set.

2014-02-27: Variable output format

The output format for time series of selected variables at selected objects is now controlled via the configuration data (new keyword `outputFormat`). Supported choices are `outputFormat=tab` and `outputFormat=json`. In the first case, the output format is TAB-separated text (as in previous versions) and the file name extension is set to '.txt'. In the second case, the output is formatted in Java Script Object Notation (see `http://www.json.org`) and the file extension is set to '.json'. The json-format is self-documenting and efficient with respect to memory consumption. There are json-plugins for many softwares and languages (for example, the R-package 'rjson'). Files in json-Format can still be imported in spreadsheet software using the ',' as column separator.

2014-02-21: Output formatting

The table used to request output for selected objects / variables must contain a 'digits' field now. The field specifies the number of digits after the decimal place to be used when printing the respective output time series. The numbers are now output in *fixed* notation rather than *scientific* notation. Debug output and state output are not affected.

2013-03-18: Computation time and multithreading

The computation time (time consumption of a model run) was calculated incorrectly if OMP parallel processing was enabled. In that case, the C++ `clock()` function returned unrealistic results. The computation time is now correctly obtained using calls to `time(0)` as an alternative. Also fixed was an error in the number of CPUs being displayed on screen and written to the log file. This number was always 1, even if multiple CPUs were actually used.

2012-07-05: Error messages in case of FPE

The diagnostic message for reporting of floating point exceptions was improved. In former version, the model stopped right after detection of the first invalid number. The revised version reports all invalid values of state variables and output variables before throwing an exception that causes the model to halt. This makes it easier to find the cause of the problem without extensive debugging.

2012-06-19: New interface of numerical ODE solver

The interface of the Runge-Kutta-based ODE solver was simplified. In addition, the length of a simulation time step `delta_t` is now passed to the method `derivsScal` as an additional argument. This allows for the conversion of external inputs which are given as time-step sum values (precipitation, for example) into rate values (m/s). The rate values can than be used in the formulation of derivatives.

2012-05-14: Single output directory + changed keyword in config

In former versions, the user was able to specify different target directories for

1. the output of selected variables of selected models,
2. debug outputs,
3. model states.

This was inconvenient in cases where a model is run repeatedly under control of another software. In such a case, the caller had to check for output in all three directories (to avoid conflicts with overwriting of files). In the revised version, the user must specify a single directory and all of the above outputs are written to that directory. It is guaranteed that the names of all files are unique (through the use of different extensions, defined in `echse_globalConst.h`).

Consequently, the keywords to be used in the config file (or at the command line) where changed. The 3 formerly used keywords `directory_selectedOutput`, `directory_debugOutput`, and `directory_stateOutput` are no longer valid. The new keyword for the single output directory is `outputDirectory`.

2012-05-02: Changed keywords in config file

The keywords related to the classes' parameter files have been modified. In previous versions, the keywords contained the class name enclosed in parenthesis. This may be problematic if the parameter files are specified at the command line (rather than in a config file). The new keywords no longer contain parenthesis (see table below).

Old keyword for class 'xxx'	**New keyword for class 'xxx'**
`table_paramsFunIndividual(xxx)`	`xxx_paramsFunIndividual`
`table_paramsFunShared(xxx)`	`xxx_paramsFunShared`
`table_paramsNumIndividual(xxx)`	`xxx_paramsNumIndividual`
`table_paramsNumShared(xxx)`	`xxx_paramsNumShared`

2012-02-28: Handling of underflow in string to double conversion

The `to_double` function from `typeconv.h` was adapted to properly handle values very close to zero. Strings that cause an underflow (like '1.0e-350', for example) are now converted to zero. An exceptions is no longer thrown.

2012-02-28: Exceptions in parallel regions

Exceptions are no longer thrown within the region of parallel processing. Instead, exceptions are registered and thrown after leaving the parallel region.

2012-02-05: Support for numerical integration of scalar state variables (built-in ODE solver)

- The 'abstractObject' class has a new virtual method called 'derivsScal'. A template of this method is created by the code generator for each child class of 'abstractObject'.
- If the simulation of the scalar state variable(s) requires a numerical solution of the underlying ODE(s), the model developer must implement this new method. Otherwise, the method's interior can remain essentially empty. The purpose of the method is to compute the derivatives of the scalar state variables with respect to time.
- In the 'simulate' method, one can make a call to the ODE solver `odesolve_nonstiff`, which resides in the new source files `echse_coreFunct_solveODE.h/.cpp`. This is a translation of the 5th order Runge-Kutta solver from Numerical recipes in Fortran 90. The ODE solver takes a pointer to the active object as input and internally calls the object's 'derivsScal' method to compute the required derivatives.
- The new data access methods `stateScal_all()` and `set_stateScal_all()` were added to the 'abstractObject' class (see PDF documentation). These methods can be used to retrieve and set the values of all scalar state variables as a whole. This is quite useful when using the numerical ODE solver.

- The new data access methods `stateScal_all()` and `set_stateScal_all()` were added to the 'abstractObject' class (see PDF documentation). These methods can be used to retrieve and set the values of all scalar state variables as a whole. This is quite useful when using the numerical ODE solver.
- The constants used by the data access functions are now held in two separate namespaces (= change implemented in the code generator). This is to make the access functions for scalar state variables accessible in the 'simulate' method only but *not* in the 'derivsScal' method. It would be an (hard to trace) error if those access functions would be called in 'derivsScal'. The proper way (and only) way to access the values of scalar states in 'derivsScal' is via the passed vector argument.

2012-02-05: Changed organization of source files

The user-supplied code for the body of a class method is now included using an `#include` directive. This allows for a better separation of user-supplied code and generated code.

2012-01-19: Saving of final model state

A logical switch has been added to enable/disable the saving of the model's final state. The new keyword in the configuration data is `saveFinalState`. If true, the final state is saved even if the time corresponding to the end of the simulation period is not listed in the file controlling the state output (defined by `table_stateOutput`). If false, the model state is only saved at the listed, pre-defined times.

The use of the new switch is especially convenient in a forecasting environment where multiple model runs are carried out and the final state of each run has to be saved. One no longer needs to adapt the file defined by `table_stateOutput` for that purpose.

2012-01-19: Passing of configuration data via the command line

Configuration data can now be specified *either* in the configuration file (as in earlier versions) *or* at the command line (which has not been possible before). Thus, it is no longer necessary to edit the configuration file if you need to call a model multiple times with variable configuration data. The syntax for command line arguments is the same as for the records in the configuration file: `key=value`. Note that it is an error to re-define the value for a key which is already defined in the configuration file (and vice versa). See the updated documentation for handling of special characters in the configuration data.

2012-01-17: Modified log and screen messages

- Progress info is only printed if 1, 2, 5, 10, 20, or 50% of the simulation is done (and 'silent' is true). These thresholds are defined in `echse_globalConst.h`. The progress info, along with an estimate of the remaining computation time is now also written to the log file.

- The return codes of external commands are now reported in the log file.

2012-01-12: Revised multi-run mode

1. All variables related to the control of the multi-run mode were put into a structure (struct) 'multiRun'.
2. The keywords in the configuration file related to multi-run mode were adapted (see table below).
3. The user must now specify the return code of the pre-run and post-run command that should be interpreted as successful termination. Typically, by convention, a return code of zero (0) indicates success.
4. The user must now also specify a return code of the post-run command which should be interpreted as an exit signal (but not as an error). This allows for exiting the multi-run loop if some condition is met. In former versions, such early termination was impossible since the number of run was fixed.

The following table lists all parameters controlling the multi-run mode, including the old and new keywords used in the configuration file.

New keyword	Old keyword	Type	Meaning
`multiRun.cmd_pre`	`command_preRun`	string	Command executed before each run.
`multiRun.cmd_post`	`command_postRun`	string	Command executed after each run.
`multiRun.returncode_pre_normal`	did not exist	integer	Return code indicating successful termination of the pre-run command (typically 0). Any other return code is interpreted as an error condition and will cause an exception.
`multiRun.returncode_post_normal`	did not exist	integer	Return code indicating successful termination of the post-run command (typically 0). A return code being equal neither to `multiRun.returncode_post_normal` nor `multiRun.returncode_post_break` is treated as an error condition.
`multiRun.returncode_post_break`	did not exist	integer	Return code indicating that the post-run command terminated successfully and the multi-run loop should be exited *now*, even if the number of runs carried out so far is less than `multiRun.max_runs`. A return code being equal neither to `multiRun.returncode_post_normal` nor `multiRun.returncode_post_break` is treated as an error condition.
`multiRun.max_runs`	`number_of_runs`	integer	The maximum number of runs to be carried out. The actual number may be lower if the post-run command issues a return code of `multiRun.returncode_post_break` before the maximum number of runs is reached.
`multiRun.reInit_paramsNum`	`reInit_paramsNum`	logical	Should object-specific scalar parameters be re-initialized before each run?
`multiRun.reInit_paramsFun`	`reInit_paramsFun`	logical	Should object-specific parameter functions be re-initialized before each run?
`multiRun.reInit_sharedParamsNum`	`reInit_sharedParamsNum`	logical	Should group-specific scalar parameters be re-initialized before each run?
`multiRun.reInit_sharedParamsFun`	`reInit_sharedParamsFun`	logical	Should group-specific parameter functions be re-initialized before each run?

2011-11-30: Changes to code generator

The code generator no longer outputs a class documentation in HTML format. It was found that the class features are better documented in the code generator's input file and there is no need to translate this to HTML.

2011-11-27: Modified interface of 'simulate' method

The argument `delta_t`, representing the simulation time step, is now an unsigned integer. It has been declared as a signed int before.

2011-11-14: Modified keywords in config file

Some keywords of the config files have been modified for consistency of identifiers (in the code) and keywords (in config file).

Old keyword	New keyword
`table_printedOutputs`	`table_selectedOutput`
`table_debugObjects`	`table_debugOutput`

2011-11-10: Consistent columns names in input files

The column names in some input files have been adapted to make them consistent in all input files. All column names are now defined in the new source file `echse_globalConst.h`. Changes to this source file will affect all input files of a model. To adapt the column names of existing input files (or preprocessing scripts producing such files), please consult the mentioned source file and the comments therein.

The required changes are (including the update from 2011-11-08):

Old colum name	New column name	Affected files
`model` or `id_model`	`object`	Various
`sourceModel`	`sourceObject`	Linkage table
`targetModel`	`targetObject`	Linkage table
`id_family`	`objectGroup`	Object declaration table
`id_function`	`function`	Parameter functions assignment tables of all objectGroups

In summary, the prefix `id_` has to be removed everywhere, `model` has to be replaced by `object` and `family` must be substituted with `objectGroup`.

2011-11-08: Changed names of identifiers (and columns names in input files)

The identifier names were changed with the intention of (1) avoiding confusion due to multiple uses of the term 'model', (2) making identifier names more meaningful (improved self-documentation of the code). The old and new names are listed below (`*` is a wildcard matching any character or nothing):

old	new
`*model*`	`*object*`
`*family*`	`*group*`

The identifier names were changed in the ECHSE core sources and the code-generator. As a consequence, the names of some columns of the input files had to be adjusted as well (accoding to the above table). Due to this fact, all utility programs (pre-processors) used for generating input files for any ECHSE-model need to be adapted so as to use the new names. In particular column names like `id_model` or `id_family` have to be substituted by `id_object` or `id_group`, respectively in the generated input files.

2011-10-27: Code security improved

With the update from 2011-10-11 index checks for vectors can optionally be turned off. This was a potential problem because, even if the index-constants supplied by the code-generator are used, the user may still write unsafe code (by accidentially using an index constant for state variables when accessing a parameter, for example). In the updated version, this is no longer possible. The index constants are now specific for a particular item type, i.e. an index constant for scalar state variables can only be used to access an element in the vector of scalar state variables, for example. Furthermore, the naming of the access constants was changed. The names of the constants are now identical with the names of the items (as used in the input to the code generator). Therefore, these names cannot be used for local variables/functions in the simulate-method (the compiler will complain, if name clashes appear).

2011-10-24: Simulation of a subset of models

In former versions, one could select a subset of models for actual simulation by providing the ID of the 'first' and 'last' model. This option does no longer exist and the corresponding keys (`model_first` and `model_final`) in the config file are no longer required/used. The reason for deleting this option is that, due to other changes to the code, the sequence in which the objects are simulated is not known in advance (it is determined by the software based on the objects input-output relations). Therefore, a user-supplied specification of the 'first' and 'last' object is not appropriate anymore. With the updated version, all models which are declared in the model declaration file are simulated. Thus, in order to restrict the simulation to only a subset of models, this file must be filtered (or one must use comments). All other input files may be left unchanged (info for 'non-declared' models is simply ignored). In comparison with the old approach of specifying a 'first' and 'last' model, there is a gain in execution speed since memory is now allocated only for those models that are actually simulated. Moreover, with the new approach, improper input-output relations are easier to detect. If a declared object 'A' (i.e. an object to be simulated) requires input from a non-simulated (i.e. undeclared) object 'B', an exception is now generated. (Before the update, the user had to be sure that 'B' is not 'inactive' because non-sense data would be exchanged in such case.)

2011-10-24: Type of input-output relations

In the new version, one has to specify for each input-output relation whether it is a 'forward' or 'backward' relation. For that purpose, the table of simulated boundary has been expanded by the column `forwardType` which is of boolean

type. Looking at a single time step, a 'forward' relation means that, if an object 'A' uses an output from an object 'B' as input, the object 'B' is simulated before object 'A'. In contrast, a 'backward' relation would mean that 'A' uses the output of 'B' despite of the fact that 'A' is simulated before 'B'. Thus, if the relation is of the forward type, 'A' gets 'new' information from 'B' (i.e. information corresponding to the end of the current time step). If the relation was of the backward type, 'A' gets 'old' information from 'B' (i.e. information corresponding to the end of the previous time step). It is important (and automatically checked) that the input-output relation of two objects is consistent if the objects exchange more than one variable. In such a case, the relations for all variables must be either 'forward' or 'backward' (mixing both types would result in a circle-reference).

2011-10-24: Sequence of object simulation

The order in which the objects appear in the declaration file has no importance anymore. The sequence in which the objects are simulated is now determined by the software based on the objects input-output relations (namely those relations being of the 'forward' type). In this contect, each object is assigned a 'level'. Generally, if object 'A' gets a 'forward' input from object 'B', the level of A is greater than the level of 'B'. It is guaranteed that all objects with a lower level are simulated before any object with a higher level. This is basically a precondition for parallelization.

2011-10-24: Parallelization

Objects being of the same level can be simulated in parallel without any effect. Parallelization has been implemented using OpenMP (shared-memory concept). The config-file has been expanded for the two following keywords:

`multithread` — Type: boolean. If true, multithreading is enabled. If false, the simulation is done in a single thread.

`singlethread_if_less_than` — Type: integer ≥ 1. With this key one can choose when multi-threading should actually be activated. If `multithread=true`, the simulation will ONLY be split into multiple thread (depending on available CPUs) if the number of objects of the same level is equal or greated than this number. If there are less objects at a particular level, these objects will be simulated sequentially rather than in parallel. If `multithread=false`, the specified number is without effect.

Tests on dual-core machines for medium-sized problems revealed that multithreading can lead to a significant slow-down rather that to the desired gain in performance. A slow-down is almost guaranteed if the value of `multithread_if_more_than` is small (or 1, which is the lower limit). The reason is that the creation of parallel-working threads and the subsequent synchronization is costly in terms of computation time. Therefore, a sequential simulation may be (much) faster, especially when there are only few objects of the same level.

It has to be tested individually for each problem and computer architecture (number of CPUs, cache, memory, ...) whether parallelization actually improves the performance.

2011-10-19: Format of debug output changed

1. The output is now a plain text table (like all other in-/output files). In earlier versions, the values of all elements of a vector state variable were printed at the same line. This led to a variable number of columns and the inspection of debug files was more difficult.

2. In addition to the values of state variables (scalar and vector), the debug output now also contains infos on inputs (external and simulated) as well as the computed output variables.

2011-10-14: Automatic documentation

The code generator now requires a basic description and a unit to be specified for all items of a class. The information is written to the generated HTML documentation. NOTE: This is no longer true (see log entry from 2011-11-30).

2011-10-11: Optimization

Range checks in the functions used to access an object's data (values of state variables, parameters, etc.) are now optional. I.e. the explicit check whether an element index for a specific vector is out of range can now be turned off. This can lead to a significant gain in performance. However, skipping the checks is dangerous if no other measures are taken to prevent the use of invalid indices (see below). If checks are turned off, the use of invalid indices may lead to runtime errors (seg. fault) at best. In worse cases, the model will complete normally but the results are non-sense since random data are used in computations. Implementation: Range checks are supressed if the preprocessor macro `CHECK_RANGE` has a value of zero (`'#define CHECK_RANGE 0'` directive). If the `CHECK_RANGE` has a non-zero value, range checks will be performed and an exception will be thrown if an index is out of range. Thus, to turn checks on/off, recompilation is necessary. The macro is defined in the new source file `echse_options.h`. Notes:

1. Do not set `CHECK_RANGE` to zero unless you understand the consequences.

2. When range checks are turned off, you should take other measures to make sure that the used indices are valid. For example, you should ONLY use the appropriate constants created by the code-generator rather than your own index variables/constants.

3. You should run your model once with range checking enabled before turning it off. After making changes to the code of the simulate() methods, you should repeat this test.

2011-10-11: Optimization

Functions to access an object's data item (state variable, parameter, etc.) are now implicit inline functions, i.e. these functions are defined inside the class.

Declaring them as `inline` outside the class doesn't work (results in undefined references in the derived classes). The inline-approach is used for the member functions of `abstractModel` (returning an objects individual data) AND `abstractModelFamily` (returning data shared by all objects of the family).

2011-10-11: Clean-up of code

Access to an object's data item (state variable, parameter, etc.) via the item name is no longer supported. In real-world applications one always wants to access an item by its index rather than its name since this is much faster. If required, the item name can still be translated into the corresponding index using the appropriate method of the model family (which is accessible through a pointer).

2011-10-07: Detection of floating point exceptions

The key `trap_fpe` was added to the config file. If set to `true`, an exception is generated if a computed value (state variable or output variable) is not a valid floating point number (i.e. NAN or INF). If set to `false`, the check is suppressed an the generated NANs or INFs are used in further computations and/or appear in output files (known as 'non-stop mode'). Note that the implemented check does not rely on signals but uses the 'isfinite()' function defined in 'cmath'. Thus, the check does not depend on the platform and/or compiler settings. The cost of trapping floating point exceptions seems to be low in normal conditions, i.e. `trap_fpe=true` will usually be the appropriate choice.

2011-10-06: Bugfix

Fixed error message in method `eval` of class `function`. Attempts to evaluate the function for invalid argument values (NAN, INF, etc.) are reported properly now.

2011-09-01: More flexible saving of model state

The model state (values of scalar and vector state variables of all models) is now output at user-selected times. The automatic output at the end of a simulation does no longer happen. The config file has a new keyword `table_stateOutput`. The value assiged to that key must be the name of a file containing a table of the desired state output times. This table should consist of a single column with name `time` (header must be present). The column should have zero or more rows, each containing a time in ISO format `YYYY-MM-DD hh:mm:ss`.

2011-08-20: Bugfix

Fixed an error in the destructor of the class `abstractModel`. The `externalInputs` are stored in a vector of vectors (1st dimension = variables, 2nd dimension = locations). Before this bug-fix, only the first dimension of this vectors-in-vector construct was removed. The second dimension was not cleaned-up properly, and the data were kept in memory. The `clear()` method for the `externalInputs`

is now also called at the very beginning of the initialization method. This guarantees that the `externalInputs` are properly updated in repeated calls of the initialization method even if the model object's destructor is not called in between. This situation arises when multiple runs are performed (model objects are instantiated/destroyed only once, `externalInputs` are updated for each run).

2011-08-19: Multi-run mode (built-in repeated execution)

It is now possible to perform multiple runs (i.e. simulations) with a single call of the executable. The idea is to save computation time in situations where many runs are required (model calibration or ensemble simulations, for example) but a large part of the model's input data is identical for each run. In such a situation it will make sense to avoid repeated reading of the same data or repeated instantiation of objects which may be re-used. The effect will be most notably when the computation time spent with actual simulation (time loop) is short in relation to the time spent with object initialization (i.e. memory allocation) and data reading. In particular, such a situation may arise in short-term ensemble prediction. The number of runs to be carried out is controlled by the new keyword `number_of_runs` which must be present in the config file. The multi-run mode is enabled when setting `number_of_runs` to a value > 1.

If the multi-run mode is to make sense, there must be a chance to modify some of the model's input data between the individual runs and possibly to save a particular run's output. The calling unit (i.e. the shell), however, cannot provide these services since control is not given back to the shell before all runs are finished. As a solution to this problem, the model itself tries to execute a user-suplied shell command before and after each model run. Such a command will usually be a call of a shell scripts or external program. This mechanism provides a great deal of flexibility (see examples below). The commands to be executed are provide via the new config file's keys `command_preRun` and `command_postRun`. The index of the current run is automatically appended as a parameter to the user-supplied command lines. This is useful for creating directory names or renaming of files, for example.

When using the multi-run mode, it is important to understand what data are constant and what data must (or may be) updated in each run. Details are given in teh following section.

Updating rules

1. **Things being constant for each run**
   - The basic objects (i.e. models and modelFamilies) remain allocated and unchanged.
   - The models' input-output relations can't change.
   - The output variables/locations remain constant.
2. **Things that are always updated before a new run**
   - Initial values of the state variables are read and set anew before each run. Re-initialization of the state variables is necessary anyway and

it is just simpler to re-read the data before each run than to keep them in memory.

- Boundary conditions are read and set anew before each run. This is necessary because there is no chance to re-use the time series data. This is due to the fact that the model does not read the data as a whole but only one/some record(s) at a time. Thus, we have no time series object im memory that could be re-used.

3. **Things being updated dependent on settings in the control file**
   The control file has four new keywords:

   - `reInit_paramsNum`
   - `reInit_paramsFun`
   - `reInit_sharedParamsNum`
   - `reInit_sharedParamsFun`

   If set to `true`, the corresponding group of parameters will be updated (read and set anew) before each model run. If set to `false`, the corresponding parameter values/functions will be re-used for all runs.

Summary of settings in the config file related to multi-run mode

`number_of_runs`	An integer > 0
`command_preRun`	An executable shell command or `-` for 'do nothing'
`command_postRun`	An executable shell command or `-` for 'do nothing'
`reInit_paramsNum`	true/false
`reInit_paramsFun`	true/false
`reInit_sharedParamsNum`	true/false
`reInit_sharedParamsFun`	true/false

How to use the multi-run mode

Model calibration One has to set one (or more) of the keys related to parameter updating (such as `reInit_paramsNum`) to a value of `true`. The external command executed before each run must organize the updating of the respective parameter files (note that the files' path is constant as defined in the config file). The external command executed after a run may be used to save the output for the current parameter set, for example. Since identical initial values and boundary conditions are usually required for each run one simply has to make sure that the respective input files are NOT modified by the external commands (i.e. no action is required).

Parameter ensembles Works just like model calibration.

Initial value ensembles All keys related to parameter updating (such as `reInit_paramsNum`) are set to a value of `false` to avoid repeated reading of the same data. The external command executed before each run must organize the updating of the files with the initial values (note that the files' path is constant as defined in the config file). Usually, the external command executed before a run will replace some files by new versions. The external command executed after a run may be used to save the output, for example. The external command(s) should NOT modify the boundary condition data.

Boundary condition ensembles Works similarly to initial value ensembles but now, the external command(s) have to modify the boundary condition files rather than the files with initial values.

2011-08-15: Optimization

Optimized conversion of strings to numeric data, in particular `as_double()` defined in `typeconv.h`. These functions now have much better performance (no instantiation of temporary stringstream object). This makes a significant difference when reading large time series files or other numeric tables.

2011-08-15: Output formatting

Format of numbers in output time series (except in files of final model state) is now consistently `%0.3e`.

2011-08-12: Parameters shared by models of a family

One can now define numerical parameters and parameter functions for a model family (instead of for individual models only). These parameters are shared by all models of the respective model family. This reduces the size of input files, the effort to read the data, as well as memory consumption in cases where the values of a parameter or parameter functions are identical for all models of a model family. The item names in the input table of the code generator are `sharedParamsNum` and `sharedParamsFun` for numerical parameters and parameter functions, respectively. The respective new keywords in the config file are `table_paramsNumShared(<family>)` and `table_paramsFunShared(<family>)`. As usual, dummy files must be referenced if no shared parameters exist for a particular model family. To avoid confusion, the previously used keywords for the model specific parameter tables were changed in the configuration file (`table_paramsNum(<family>)` is now `table_paramsNumIndividual(<family>)` and `table_paramsFun(<family>)` is now `table_paramsFunIndividual(<family>)`).

2011-08-05: Optimization attempt

Attempt to further increase execution speed by modifying the vector-range check in the data access methods (like `paramNum(const unsigned int index)` for example): Instead of checking the supplied index against the value of `theDataVector.size()`, it was checked against a stored size value (which was implemented as a member of the model family). The gain in speed proved to be negligible however. Obviously, the calls to the `size()`-function are very efficient and the access to the stored value (required resolving 1 pointer + 1 function call) was not faster. Consequently, the modifications were not adopted (for the sake of simpler code). If execution speed does really matter, one might consider to comment out the range-checks (after the specific user-supplied code has been tested thoroughly).

2011-08-04: Improved traceback info and performance

Replaced method names for error messages (formerly const strings) by macro `__PRETTY_FUNCTION__` in the entire cpp-code. This avoids the instantiation of

string objects in the method's code at runtime (which severely affects performance) and facilitates code maintenance.

2011-08-01: Output of traceback info

`'file_err'` is now an additional command line argument. In case of an exception, the traceback info will be written to the specified file. The format of the traceback file is controlled by the command line argument `'format_err'`. Possible choices for `'format_err'` include `'xml'` and `'html'`. Any other value will cause the output to be in simple tab-separated text format. If writing of the traceback file fails (for example, because the specified path/file is invalid) the traceback will be send to screen. The recommended way of testing for successful execution is still to check the return code (0 if OK, <> 0 in case of an early stop). The traceback file is opened in append-mode, i.e. an existing file will NOT be overwritten. Therefore, if the file already exists, the latest error will be found at the end of the file. In most instances you want to remove an old traceback file before running the model.