Development of new solvers

Introduction

There is a possibility to develop own external modules (called solvers) in the Dyssol. The functionality of these modules can be afterwards used in all units by adding them as unit parameters (refer to 'BaseUnit.pdf' AddSolver*()).

In the current version only one type of solvers available: agglomeration solver. Basically, all solvers have a set of constant functions and parameters, which are available in each new solver (refer to 'ExternalSolver.pdf'), and a set of specific ones, which depend on the solver's type. New types of solvers can be added by request and will include a set of parameters and functions that are needed to solve a specific problem.

There is a possibility to implement several solvers of one type (e.g. with different models) and then choose a specific one to use it in unit using user interface, section 'Unit parameters'.

To start development of solvers the following conditions must be previously satisfied (refer to 'Configuration of VCProject.pdf'):

1. Installed Microsoft Visual Studio 2015 (Community).
2. Configured template project VCProject.

1. Adding new solver to the template project

1. Copy the desired template of the unit from *<PathToSolution>\VCProject\SolversTemplates* to the folder *Solvers* in solution (*<PathToSolution>\VCProject\Solvers*).
2. Rename template's folder according to the name of your new solver (further: ***<MySolverFolder>***). The name can be chosen freely.
3. Rename project files in template's folder (**.vcxproj*, **.vcxproj.filters*) according to the name of the new solver.
4. Run the solution file (*<PathToSolution>\Dyssol.sln*) to open it in Visual Studio.
5. Add project with your new solver to the solution. To do this, select in Visual Studio [*File → Add → Existing Project*] and specify path to the project file (*<PathToSolution>\VCProject\Solvers\<MySolverFolder>\<*.vcxproj>*).
6. Rename added project in Visual Studio according to the name of your solver.

Now one can implement functionality of new solver. The list of available functions depends on type of selected solver.

To build your solution press F7, to run it in debug mode press F5. Files with new solvers will be placed to *<PathToSolution>\VCProject\Debug*.

As debug versions of compiled and built solvers contain a lot of additional information, which is used by Visual Studio to perform debugging, their calculation efficiency can be dramatically low. Thus, for the simulation purposes solvers should be built in Release mode:

2. Configuring Dyssol to work with implemented solvers

1. Build your solvers in release mode. To do this open your solution in Visual Studio (run file *<PathToSolution>\VCProject.sln*), switch *Solution configuration* combo box from the toolbox of Visual Studio from *Debug* to *Release* and build the project (press F7 or choose [*Build → Build project*] in program menu).
2. Configure Dyssol by adding the path to new solvers: run Dyssol, choose [Tools → *Options → Model manager*] and add path to your solvers (*<PathToSolution>\VCProject\Release*).

Now all new developed units will be available in *Dyssol*.

In general, usual configuration of *Model manager* should include following path for solvers:

- *<InstallationPath>\Solvers* - list of standard solvers;
- *<PathToSolution>\VCProject\SolversDebugLibs* – debug versions of standard solvers;
- *<PathToSolution>\VCProject\Debug* – debug versions of developed solvers;
- *<PathToSolution>\VCProject\Release – release versions of developed solvers.*

3. Development of agglomeration solver

***Solver*::*Solver* ()**
Constructor of the solver: called only once when solver is added to the unit. In this function a set of parameters should be specified:

1. Basic info:
- *m_solverName* – Name of the solver that will be displayed in Dyssol.
- *m_authorName* – Solver's author
- *m_solverUniqueKey* – Unique identificator of the solver. Simulation environment distinguishes different solvers with the help of this identificator. You must ensure that ID of your solver is unique. This ID can be created manually or using GUID-generator of Visual Studio (*Tools->GUID Genarator*).
2. All operations, which should take place only once during the solver's creation.

Solver*::*~Solver ()
Destructor of the solver: called only once when solver is removed from the unit. Here all memory which has been previously allocated in the constructor should be freed.

Solver::Initialize (vector<double> grid, double betta0, EKernels kernel, size_t rank, vector<double> params)
Solver‘s initialization. This function is called only once for each simulation during the initialization of unit. All operations, which should take place only once after the solver's creation should be implemented here. Implementation of this function is not obligatory and can be skipped.

Solver::Calculate (vector<double> N, vector<double> BRate, vector<double> DRate)
Calculates birth and death rates depending on particle size distribution. All logic of the solver must be implemented here.

Solver::Finalize ()
Unit‘s finalization. This function is called only once for each simulation during the finalization of unti. Here one can perform closing and cleaning operations to prepare for the next possible simulation run. Implementation of this function is not obligatory and can be skipped.

4. Configuring solver to work with MATLAB

There is a possibility to use MATLAB Engine API in the Dyssol during the development of solvers. It requires an installed **32-bit version** of MATLAB. For API description refer to following links:

- http://de.mathworks.com/help/matlab/calling-matlab-engine-from-c-c-and-fortran-programs.html
- http://de.mathworks.com/help/matlab/cc-mx-matrix-library.html

To enable interaction with MATLAB configure template project with your solver:

1. Add a new environment variable in Windows with the path to the MATLAB installation directory: *Computer → Properties → Advanced system settings → Environment variables → System variables → New* → Variable Name: *MATLAB_PATH*; Variable value: path to installed 32-bit version of MATLAB (e.g. *C:\Program Files (x86)\MATLAB\R2014b*). It may require restarting the Visual Studio or computer to apply changes.
2. Provide the main project of template solution with path to MATLAB libraries: select project *ModelsAPI* in solution explorer, then choose [*Project → Properties → Configuration Properties → Environment*], set combo box *Configuration* in the top of the window to position *All Configurations* and provide the *Environment* field with parameter *PATH=$(MATLAB_PATH)\bin\win32*.
3. Provide solver's project with the path to MATLAB libraries: select project with your solver in solution explorer, then choose [*Project → Properties → Configuration Properties → Environment*], set combo box *Configuration* in the top of the window to position *All Configurations* and provide the *Environment* field with parameter *PATH=$(MATLAB_PATH)\bin\win32*.
4. Add MATLAB libraries to the solver's project: select project with your solver in solution explorer, then choose [*Project → Properties → Configuration Properties → Linker → Input → Additional Dependencies*], set combo box *Configuration* in the top of the window to position *All Configurations* and add following four libraries at the beginning of the input field: *libmx.lib;libmat.lib;libeng.lib;libmex.lib;*
5. Insert MATLAB's header in Solver.h: add the line `#include "engine.h"` to the include section at the top of your Solver.h file.