OMSimulator Documentation

Release v2.1.0-dev-108-g6d1ad54

Lennart Ochel

May 20, 2019

CONTENTS

CHAPTER

ONE

INTRODUCTION

The OMSimulator project is a FMI-based co-simulation tool that supports ordinary (i.e., non-delayed) and TLM connections.

CHAPTER

TWO

OMSIMULATOR

OMSimulator is a command line wrapper for the OMSimulatorLib library.

2.1 OMSimulator Flags

A brief description of all command line flags will be displayed using `OMSimulator --help`:

```
info:    Usage: OMSimulator [Options] [Lua script] [FMU] [SSP file]
         Options:
           --clearAllOptions                Reset all flags to default␣
↪values
           --fetchAllVars=<arg>             Workaround for certain FMUs␣
↪that do not update all internal dependencies automatically
           --help [-h]                      Displays the help text
           --ignoreInitialUnknowns=<bool>   Ignore the initial unknowns␣
↪from the modelDescription.xml
           --inputExtrapolation=<bool>      Enables input extrapolation␣
↪using derivative information
           --intervals=<int> [-i]           Specifies the number of␣
↪communication points (arg > 1)
           --logFile=<arg> [-l]             Specifies the logfile (stdout␣
↪is used if no log file is specified)
           --logLevel=<int>                 0 default, 1 debug, 2␣
↪debug+trace
           --mode=<arg> [-m]                Forces a certain FMI mode iff␣
↪the FMU provides cs and me ([cs], me)
           --numProcs=<int> [-n]            Specifies the max. number of␣
↪processors to use (0=auto, 1=default)
           --progressBar=<bool>             Shows a progress bar for the␣
↪simulation progress in the terminal
           --realTime=<bool>                Experimental feature for (soft)␣
↪real-time co-simulation
           --resultFile=<arg> [-r]          Specifies the name of the␣
↪output result file
           --setInputDerivatives=<bool>     Deprecated; see '--
↪inputExtrapolation'
           --solver=<arg>                   Specifies the integration␣
↪method (euler, [cvode])
           --solverStats=<bool>             Adds solver stats to the result␣
↪file, e.g. step size (not supported for all solvers)
           --startTime=<double> [-s]        Specifies the start time
           --stopTime=<double> [-t]         Specifies the stop time
           --stripRoot=<bool>               Removes the root system prefix␣
↪from all exported signals
```

(continues on next page)

(continued from previous page)

```
          --suppressPath=<bool>            Supresses path information in␣
↪info messages; especially useful for testing
          --tempDir=<arg>                  Specifies the temp directory
          --timeout=<int>                  Specifies the maximum allowed␣
↪time in seconds for running a simulation (0 disables)
          --tolerance=<double>             Specifies the relative tolerance
          --version [-v]                   Displays version information
          --wallTime=<bool>                Add wall time information for␣
↪to the result file
          --workingDir=<arg>               Specifies the working directory
```

2.2 Examples

```
OMSimulator --timeout 180 example.lua
```

CHAPTER

THREE

OMSIMULATORLIB

This library is the core of OMSimulator and provides a C interface that can easily be utilized to handle co-simulation scenarios.

3.1 C-API

3.1.1 RunFile

Simulates a single FMU or SSP model.

```
oms_status_enu_t oms_RunFile(const char* filename);
```

3.1.2 addBus

Adds a bus to a given component.

```
oms_status_enu_t oms_addBus(const char* cref);
```

3.1.3 addConnection

Adds a new connection between connectors *A* and *B*. The connectors need to be specified as fully qualified component references, e.g., *"model.system.component.signal"*.

```
oms_status_enu_t oms_addConnection(const char* crefA, const char* crefB);
```

The two arguments *crefA* and *crefB* get swapped automatically if necessary.

3.1.4 addConnector

Adds a connector to a given component.

```
oms_status_enu_t oms_addConnector(const char* cref, oms_causality_enu_t␣
↪causality, oms_signal_type_enu_t type);
```

3.1.5 addConnectorToBus

Adds a connector to a bus.

```
oms_status_enu_t oms_addConnectorToBus(const char* busCref, const char*␣
↪connectorCref);
```

3.1.6 addConnectorToTLMBus

Adds a connector to a TLM bus.

```
oms_status_enu_t oms_addConnectorToTLMBus(const char* busCref, const char*␣
↪connectorCref, const char *type);
```

3.1.7 addExternalModel

Adds an external model to a TLM system.

```
oms_status_enu_t oms_addExternalModel(const char* cref, const char* path,␣
↪const char* startscript);
```

3.1.8 addSignalsToResults

Add all variables that match the given regex to the result file.

```
oms_status_enu_t oms_addSignalsToResults(const char* cref, const char*␣
↪regex);
```

The second argument, i.e. regex, is considered as a regular expression (C++11). ".*" and "*(.)**" can be used to hit all variables.

3.1.9 addSubModel

Adds a component to a system.

```
oms_status_enu_t oms_addSubModel(const char* cref, const char* fmuPath);
```

3.1.10 addSystem

Adds a (sub-)system to a model or system.

```
oms_status_enu_t oms_addSystem(const char* cref, oms_system_enu_t type);
```

3.1.11 addTLMBus

Adds a TLM bus.

```
oms_status_enu_t oms_addTLMBus(const char* cref, oms_tlm_domain_t domain,␣
↪const int dimensions, const oms_tlm_interpolation_t interpolation);
```

3.1.12 addTLMConnection

Connects two TLM connectors.

```
oms_status_enu_t oms_addTLMConnection(const char* crefA, const char* crefB,
↪ double delay, double alpha, double linearimpedance, double␣
↪angularimpedance);
```

3.1.13 cancelSimulation_asynchronous

Cancels a running asynchronous simulation.

```
oms_status_enu_t oms_cancelSimulation_asynchronous(const char* cref);
```

3.1.14 compareSimulationResults

This function compares a given signal of two result files within absolute and relative tolerances.

```
int oms_compareSimulationResults(const char* filenameA, const char*␣
↪filenameB, const char* var, double relTol, double absTol);
```

3.1.15 copySystem

Copies a system.

```
oms_status_enu_t oms_copySystem(const char* source, const char* target);
```

3.1.16 delete

Deletes a connector, component, system, or model object.

```
oms_status_enu_t oms_delete(const char* cref);
```

3.1.17 deleteConnection

Deletes the connection between connectors *crefA* and *crefB*.

```
oms_status_enu_t oms_deleteConnection(const char* crefA, const char*␣
↪crefB);
```

The two arguments *crefA* and *crefB* get swapped automatically if necessary.

3.1.18 deleteConnectorFromBus

Deletes a connector from a given bus.

```
oms_status_enu_t oms_deleteConnectorFromBus(const char* busCref, const␣
↪char* connectorCref);
```

3.1.19 deleteConnectorFromTLMBus

Deletes a connector from a given TLM bus.

```
oms_status_enu_t oms_deleteConnectorFromTLMBus(const char* busCref, const␣
↪char* connectorCref);
```

3.1.20 export

Exports a composite model to a SPP file.

```
oms_status_enu_t oms_export(const char* cref, const char* filename);
```

3.1.21 exportDependencyGraphs

Export the dependency graphs of a given model to dot files.

```
oms_status_enu_t oms_exportDependencyGraphs(const char* cref, const char*␣
↪initialization, const char* simulation);
```

3.1.22 extractFMIKind

Extracts the FMI kind of a given FMU from the file system.

```
oms_status_enu_t oms_extractFMIKind(const char* filename, oms_fmi_kind_enu_
↪t* kind);
```

3.1.23 freeMemory

Free the memory allocated by some other API. Pass the object for which memory is allocated.

```
void oms_freeMemory(void* obj);
```

3.1.24 getBoolean

Get boolean value of given signal.

```
oms_status_enu_t oms_getBoolean(const char* cref, bool* value);
```

3.1.25 getBus

Gets the bus object.

```
oms_status_enu_t oms_getBus(const char* cref, oms_busconnector_t**␣
↪busConnector);
```

3.1.26 getComponentType

Gets the type of the given component.

```
oms_status_enu_t oms_getComponentType(const char* cref, oms_component_enu_
↪t* type);
```

3.1.27 getConnections

Get list of all connections from a given component.

```
oms_status_enu_t oms_getConnections(const char* cref, oms_connection_t***␣
↪connections);
```

3.1.28 getConnector

Gets the connector object of the given connector cref.

```
oms_status_enu_t oms_getConnector(const char* cref, oms_connector_t**␣
↪connector);
```

3.1.29 getElement

Get element information of a given component reference.

```
oms_status_enu_t oms_getElement(const char* cref, oms_element_t** element);
```

3.1.30 getElements

Get list of all sub-components of a given component reference.

```
oms_status_enu_t oms_getElements(const char* cref, oms_element_t***␣
↪elements);
```

3.1.31 getFMUInfo

Returns FMU specific information.

```
oms_status_enu_t oms_getFMUInfo(const char* cref, const oms_fmu_info_t**␣
↪fmuInfo);
```

3.1.32 getFixedStepSize

Gets the fixed step size. Can be used for the communication step size of co-simulation systems and also for the integrator step size in model exchange systems.

```
oms_status_enu_t oms_getFixedStepSize(const char* cref, double* stepSize);
```

3.1.33 getInteger

Get integer value of given signal.

```
oms_status_enu_t oms_getInteger(const char* cref, int* value);
```

3.1.34 getModelState

Gets the model state of the given model cref.

```
oms_status_enu_t oms_getModelState(const char* cref, oms_modelState_enu_t*␣
↪modelState);
```

3.1.35 getReal

Get real value.

```
oms_status_enu_t oms_getReal(const char* cref, double* value);
```

3.1.36 getSolver

Gets the selected solver method of the given system.

```
oms_status_enu_t oms_getSolver(const char* cref, oms_solver_enu_t* solver);
```

3.1.37 getStartTime

Get the start time from the model.

```
oms_status_enu_t oms_getStartTime(const char* cref, double* startTime);
```

3.1.38 getStopTime

Get the stop time from the model.

```
oms_status_enu_t oms_getStopTime(const char* cref, double* stopTime);
```

3.1.39 getSubModelPath

Returns the path of a given component.

```
oms_status_enu_t oms_getSubModelPath(const char* cref, char** path);
```

3.1.40 getSystemType

Gets the type of the given system.

```
oms_status_enu_t oms_getSystemType(const char* cref, oms_system_enu_t*␣
↪type);
```

3.1.41 getTLMBus

Gets the TLM bus objects of the given TLM bus cref.

```
oms_status_enu_t oms_getTLMBus(const char* cref, oms_tlmbusconnector_t**␣
↪tlmBusConnector);
```

3.1.42 getTLMVariableTypes

Gets the type of an TLM variable.

```
oms_status_enu_t oms_getTLMVariableTypes(oms_tlm_domain_t domain, const␣
↪int dimensions, const oms_tlm_interpolation_t interpolation, char␣
↪***types, char ***descriptions);
```

3.1.43 getTolerance

Gets the tolerance of a given system or component.

```
oms_status_enu_t oms_getTolerance(const char* cref, double*␣
↪absoluteTolerance, double* relativeTolerance);
```

3.1.44 getVariableStepSize

Gets the step size parameters.

```
oms_status_enu_t oms_getVariableStepSize(const char* cref, double*␣
↪initialStepSize, double* minimumStepSize, double* maximumStepSize);
```

3.1.45 getVersion

Returns the library's version string.

```
const char* oms_getVersion();
```

3.1.46 importFile

Imports a composite model from a SSP file.

```
oms_status_enu_t oms_importFile(const char* filename, char** cref);
```

3.1.47 importString

Imports a composite model from a string.

```
oms_status_enu_t oms_importString(const char* contents, char** cref);
```

3.1.48 initialize

Initializes a composite model.

```
oms_status_enu_t oms_initialize(const char* cref);
```

3.1.49 instantiate

Instantiates a given composite model.

```
oms_status_enu_t oms_instantiate(const char* cref);
```

3.1.50 list

Lists the SSD representation of a given model, system, or component.

Memory is allocated for *contents*. The caller is responsible to free it using the C-API. The Lua and Python bindings take care of the memory and the caller doesn't need to call free.

```
oms_status_enu_t oms_list(const char* cref, char** contents);
```

3.1.51 listUnconnectedConnectors

Lists all unconnected connectors of a given system.

Memory is allocated for *contents*. The caller is responsible to free it using the C-API. The Lua and Python bindings take care of the memory and the caller doesn't need to call free.

```
oms_status_enu_t oms_listUnconnectedConnectors(const char* cref, char**␣
↪contents);
```

3.1.52 newModel

Creates a new and yet empty composite model.

```
oms_status_enu_t oms_newModel(const char* cref);
```

3.1.53 parseModelName

Parses the model name from a given SSD representation.

Memory is allocated for *ident*. The caller is responsible to free it using the C-API. The Lua and Python bindings take care of the memory and the caller doesn't need to call free.

```
oms_status_enu_t oms_parseModelName(const char* contents, char** cref);
```

3.1.54 removeSignalsFromResults

Removes all variables that match the given regex to the result file.

```
oms_status_enu_t oms_removeSignalsFromResults(const char* cref, const␣
↪char* regex);
```

The second argument, i.e. regex, is considered as a regular expression (C++11). ".*" and *"(.)*"* can be used to hit all variables.

3.1.55 rename

Renames a model, system, or component.

```
oms_status_enu_t oms_rename(const char* cref, const char* newCref);
```

3.1.56 reset

Reset the composite model after a simulation run.

The FMUs go into the same state as after instantiation.

```
oms_status_enu_t oms_reset(const char* cref);
```

3.1.57 setActivationRatio

Experimental feature for setting the activation ratio of FMUs for experimenting with multi-rate master algorithms.

```
oms_status_enu_t experimental_setActivationRatio(const char* cref, int k);
```

3.1.58 setBoolean

Sets the value of a given boolean signal.

```
oms_status_enu_t oms_setBoolean(const char* cref, bool value);
```

3.1.59 setBusGeometry

```
oms_status_enu_t oms_setBusGeometry(const char* bus, const ssd_connector_
↪geometry_t* geometry);
```

3.1.60 setCommandLineOption

Sets special flags.

```
oms_status_enu_t oms_setCommandLineOption(const char* cmd);
```

Available flags:

```
info:    Usage: OMSimulator [Options] [Lua script] [FMU] [SSP file]
         Options:
           --clearAllOptions               Reset all flags to default␣
↪values
           --fetchAllVars=<arg>            Workaround for certain FMUs␣
↪that do not update all internal dependencies automatically
           --help [-h]                     Displays the help text
           --ignoreInitialUnknowns=<bool>  Ignore the initial unknowns␣
↪from the modelDescription.xml
           --inputExtrapolation=<bool>     Enables input extrapolation␣
↪using derivative information
           --intervals=<int> [-i]          Specifies the number of␣
↪communication points (arg > 1)
           --logFile=<arg> [-l]            Specifies the logfile (stdout␣
↪is used if no log file is specified)
           --logLevel=<int>                0 default, 1 debug, 2␣
↪debug+trace
           --mode=<arg> [-m]               Forces a certain FMI mode iff␣
↪the FMU provides cs and me ([cs], me)
           --numProcs=<int> [-n]           Specifies the max. number of␣
↪processors to use (0=auto, 1=default)
           --progressBar=<bool>            Shows a progress bar for the␣
↪simulation progress in the terminal
           --realTime=<bool>               Experimental feature for (soft)␣
↪real-time co-simulation
           --resultFile=<arg> [-r]         Specifies the name of the␣
↪output result file
           --setInputDerivatives=<bool>    Deprecated; see '--
↪inputExtrapolation'
           --solver=<arg>                  Specifies the integration␣
↪method (euler, [cvode])
           --solverStats=<bool>            Adds solver stats to the result␣
↪file, e.g. step size (not supported for all solvers)
           --startTime=<double> [-s]       Specifies the start time
```

(continues on next page)

(continued from previous page)

```
          --stopTime=<double> [-t]        Specifies the stop time
          --stripRoot=<bool>              Removes the root system prefix␣
↪from all exported signals
          --suppressPath=<bool>           Supresses path information in␣
↪info messages; especially useful for testing
          --tempDir=<arg>                 Specifies the temp directory
          --timeout=<int>                 Specifies the maximum allowed␣
↪time in seconds for running a simulation (0 disables)
          --tolerance=<double>            Specifies the relative tolerance
          --version [-v]                  Displays version information
          --wallTime=<bool>               Add wall time information for␣
↪to the result file
          --workingDir=<arg>              Specifies the working directory
```

3.1.61 setConnectionGeometry

```
oms_status_enu_t oms_setConnectionGeometry(const char* crefA, const char*␣
↪crefB, const ssd_connection_geometry_t* geometry);
```

3.1.62 setConnectorGeometry

Set geometry information to a given connector.

```
oms_status_enu_t oms_setConnectorGeometry(const char* cref, const ssd_
↪connector_geometry_t* geometry);
```

3.1.63 setElementGeometry

Set geometry information to a given component.

```
oms_status_enu_t oms_setElementGeometry(const char* cref, const ssd_
↪element_geometry_t* geometry);
```

3.1.64 setFixedStepSize

Sets the fixed step size. Can be used for the communication step size of co-simulation systems and also for the integrator step size in model exchange systems.

```
oms_status_enu_t oms_setFixedStepSize(const char* cref, double stepSize);
```

3.1.65 setInteger

Sets the value of a given integer signal.

```
oms_status_enu_t oms_setInteger(const char* cref, int value);
```

3.1.66 setLogFile

Redirects logging output to file or std streams. The warning/error counters are reset.

filename="" to redirect to std streams and proper filename to redirect to file.

```
oms_status_enu_t oms_setLogFile(const char* filename);
```

3.1.67 setLoggingCallback

Sets a callback function for the logging system.

```
void oms_setLoggingCallback(void (*cb)(oms_message_type_enu_t type, const␣
↪char* message));
```

3.1.68 setLoggingInterval

Set the logging interval of the simulation.

```
oms_status_enu_t oms_setLoggingInterval(const char* cref, double␣
↪loggingInterval);
```

3.1.69 setLoggingLevel

Enables/Disables debug logging (logDebug and logTrace).

0 default, 1 default+debug, 2 default+debug+trace

```
void oms_setLoggingLevel(int logLevel);
```

3.1.70 setMaxLogFileSize

Sets maximum log file size in MB. If the file exceeds this limit, the logging will continue on stdout.

```
void oms_setMaxLogFileSize(const unsigned long size);
```

3.1.71 setReal

Sets the value of a given real signal.

```
oms_status_enu_t oms_setReal(const char* cref, double value);
```

This function can be called in different model states:

- Before instantiation: *setReal* can be used to set start values or to define initial unknowns (e.g. parameters, states). The values are not immediately applied to the simulation unit, since it isn't actually instantiated.
- After instantiation and before initialization: Same as before instantiation, but the values are applied immediately to the simulation unit.

- After initialization: Can be used to force external inputs, which might cause discrete changes of continuous signals.

3.1.72 setRealInputDerivative

Sets the first order derivative of a real input signal.

This can only be used for CS-FMU real input signals.

```
oms_status_enu_t oms_setRealInputDerivative(const char* cref, double␣
↪value);
```

3.1.73 setResultFile

Set the result file of the simulation.

```
oms_status_enu_t oms_setResultFile(const char* cref, const char* filename,␣
↪int bufferSize);
```

The creation of a result file is omitted if the filename is an empty string.

3.1.74 setSignalFilter

```
oms_status_enu_t oms_setSignalFilter(const char* cref, const char* regex);
```

3.1.75 setSolver

Sets the solver method for the given system.

```
oms_status_enu_t oms_setSolver(const char* cref, oms_solver_enu_t solver);
```

3.1.76 setStartTime

Set the start time of the simulation.

```
oms_status_enu_t oms_setStartTime(const char* cref, double startTime);
```

3.1.77 setStopTime

Set the stop time of the simulation.

```
oms_status_enu_t oms_setStopTime(const char* cref, double stopTime);
```

3.1.78 setTLMBusGeometry

```
oms_status_enu_t oms_setTLMBusGeometry(const char* bus, const ssd_
↪connector_geometry_t* geometry);
```

3.1.79 setTLMConnectionParameters

Simulates a composite model in its own thread.

```
oms_status_enu_t oms_setTLMConnectionParameters(const char* crefA, const␣
↪char* crefB, const oms_tlm_connection_parameters_t* parameters);
```

3.1.80 setTLMPositionAndOrientation

Sets initial position and orientation for a TLM 3D interface.

```
oms_status_enu_t oms_setTLMPositionAndOrientation(cref, x1, x2, x3, A11,␣
↪A12, A13, A21, A22, A23, A31, A32, A33);
```

3.1.81 setTLMSocketData

Sets data for TLM socket communication.

```
oms_status_enu_t oms_setTLMSocketData(const char* cref, const char*␣
↪address, int managerPort, int monitorPort);
```

3.1.82 setTempDirectory

Set new temp directory.

```
oms_status_enu_t oms_setTempDirectory(const char* newTempDir);
```

3.1.83 setTolerance

Sets the tolerance for a given model or system.

```
oms_status_enu_t oms_setTolerance(const char* cref, double␣
↪absoluteTolerance, double relativeTolerance);
```

Default values are *1e-4* for both relative and absolute tolerances.

A tolerance specified for a model is automatically applied to its root system, i.e. both calls do exactly the same:

```
oms_setTolerance("model", absoluteTolerance, relativeTolerance);
oms_setTolerance("model.root", absoluteTolerance, relativeTolerance);
```

Component, e.g. FMUs, pick up the tolerances from there system. That means it is not possible to define different tolerances for FMUs in the same system right now.

In a strongly coupled system (*oms_system_sc*), the relative tolerance is used for CVODE and the absolute tolerance is used to solve algebraic loops.

In a weakly coupled system (*oms_system_wc*), both the relative and absolute tolerances are used for the adaptive step master algorithms and the absolute tolerance is used to solve algebraic loops.

3.1.84 setVariableStepSize

Sets the step size parameters for methods with stepsize control.

```
oms_status_enu_t oms_getVariableStepSize(const char* cref, double*␣
↪initialStepSize, double* minimumStepSize, double* maximumStepSize);
```

3.1.85 setWorkingDirectory

Set a new working directory.

```
oms_status_enu_t oms_setWorkingDirectory(const char* newWorkingDir);
```

3.1.86 simulate

Simulates a composite model.

```
oms_status_enu_t oms_simulate(const char* cref);
```

3.1.87 simulate_asynchronous

Simulates a composite model in its own thread.

```
oms_status_enu_t oms_simulate_asynchronous(const char* cref, void␣
↪(*cb)(const char* cref, double time, oms_status_enu_t status));
```

3.1.88 simulate_realtime

Experimental feature for (soft) real-time simulation.

```
oms_status_enu_t experimental_simulate_realtime(const char* ident);
```

3.1.89 stepUntil

Simulates a composite model until a given time value.

```
oms_status_enu_t oms_stepUntil(const char* cref, double stopTime);
```

3.1.90 terminate

Terminates a given composite model.

```
oms_status_enu_t oms_terminate(const char* cref);
```

CHAPTER

FOUR

OMSIMULATORLUA

This is a shared library that provides a Lua interface for the OMSimulatorLib library.

4.1 Examples

```
oms_setTempDirectory("./temp/")
oms_newModel("model")
oms_addSystem("model.root", oms_system_sc)

-- instantiate FMUs
oms_addSubModel("model.root.system1", "FMUs/System1.fmu")
oms_addSubModel("model.root.system2", "FMUs/System2.fmu")

-- add connections
oms_addConnection("model.root.system1.y", "model.root.system2.u")
oms_addConnection("model.root.system2.y", "model.root.system1.u")

-- simulation settings
oms_setResultFile("model", "results.mat")
oms_setStopTime("model", 0.1)
oms_setFixedStepSize("model.root", 1e-4)

oms_instantiate("model")
oms_setReal("model.root.system1.x_start", 2.5)

oms_initialize("model")
oms_simulate("model")
oms_terminate("model")
oms_delete("model")
```

4.2 Lua Scripting Commands

4.2.1 RunFile

Simulates a single FMU or SSP model.

```
# not available
```

4.2.2 addBus

Adds a bus to a given component.

```
status = oms_addBus(cref)
```

4.2.3 addConnection

Adds a new connection between connectors *A* and *B*. The connectors need to be specified as fully qualified component references, e.g., *"model.system.component.signal"*.

```
status = oms_addConnection(crefA, crefB)
```

The two arguments *crefA* and *crefB* get swapped automatically if necessary.

4.2.4 addConnector

Adds a connector to a given component.

```
status = oms_addConnector(cref, causality, type)
```

4.2.5 addConnectorToBus

Adds a connector to a bus.

```
status = oms_addConnectorToBus(busCref, connectorCref)
```

4.2.6 addConnectorToTLMBus

Adds a connector to a TLM bus.

```
status = oms_addConnectorToTLMBus(busCref, connectorCref, type)
```

4.2.7 addExternalModel

Adds an external model to a TLM system.

```
status = oms_addExternalModel(cref, path, startscript)
```

4.2.8 addSignalsToResults

Add all variables that match the given regex to the result file.

```
status = oms_addSignalsToResults(cref, regex)
```

The second argument, i.e. regex, is considered as a regular expression (C++11). *".*"* and *"(.)*"* can be used to hit all variables.

4.2.9 addSubModel

Adds a component to a system.

```
status = oms_addSubModel(cref, fmuPath)
```

4.2.10 addSystem

Adds a (sub-)system to a model or system.

```
status = oms_addSystem(cref, type)
```

4.2.11 addTLMBus

Adds a TLM bus.

```
status = oms_addTLMBus(cref, domain, dimensions, interpolation)
```

4.2.12 addTLMConnection

Connects two TLM connectors.

```
status = oms_addTLMConnection(crefA, crefB, delay, alpha, linearimpedance,␣
↪angularimpedance)
```

4.2.13 cancelSimulation_asynchronous

Cancels a running asynchronous simulation.

```
# not available
```

4.2.14 compareSimulationResults

This function compares a given signal of two result files within absolute and relative tolerances.

```
oms_compareSimulationResults(filenameA, filenameB, var, relTol, absTol)
```

The following table describes the input values:

Input	Type	Description
filenameA	String	Name of first result file to compare.
filenameB	String	Name of second result file to compare.
var	String	Name of signal to compare.
relTol	Number	Relative tolerance.
absTol	Number	Absolute tolerance.

The following table describes the return values:

Type	Description
Integer	1 if the signal is considered as equal, 0 otherwise

4.2.15 copySystem

Copies a system.

```
status = oms_copySystem(source, target)
```

4.2.16 delete

Deletes a connector, component, system, or model object.

```
status = oms_delete(cref)
```

4.2.17 deleteConnection

Deletes the connection between connectors *crefA* and *crefB*.

```
status = oms_deleteConnection(crefA, crefB)
```

The two arguments *crefA* and *crefB* get swapped automatically if necessary.

4.2.18 deleteConnectorFromBus

Deletes a connector from a given bus.

```
status = oms_deleteConnectorFromBus(busCref, connectorCref)
```

4.2.19 deleteConnectorFromTLMBus

Deletes a connector from a given TLM bus.

```
status = oms_deleteConnectorFromTLMBus(busCref, connectorCref)
```

4.2.20 export

Exports a composite model to a SPP file.

```
status = oms_export(cref, filename)
```

4.2.21 exportDependencyGraphs

Export the dependency graphs of a given model to dot files.

```
status = oms_exportDependencyGraphs(cref, initialization, simulation)
```

4.2.22 extractFMIKind

Extracts the FMI kind of a given FMU from the file system.

```
# not available
```

4.2.23 freeMemory

Free the memory allocated by some other API. Pass the object for which memory is allocated.

```
oms_freeMemory(obj)
```

4.2.24 getBoolean

Get boolean value of given signal.

```
value, status = oms_getBoolean(cref)
```

4.2.25 getBus

Gets the bus object.

```
# not available
```

4.2.26 getComponentType

Gets the type of the given component.

```
# not available
```

4.2.27 getConnections

Get list of all connections from a given component.

```
# not available
```

4.2.28 getConnector

Gets the connector object of the given connector cref.

```
# not available
```

4.2.29 getElement

Get element information of a given component reference.

```
# not available
```

4.2.30 getElements

Get list of all sub-components of a given component reference.

```
# not available
```

4.2.31 getFMUInfo

Returns FMU specific information.

```
# not available
```

4.2.32 getFixedStepSize

Gets the fixed step size. Can be used for the communication step size of co-simulation systems and also for the integrator step size in model exchange systems.

```
stepSize, status = oms_setFixedStepSize(cref)
```

4.2.33 getInteger

Get integer value of given signal.

```
value, status = oms_getInteger(cref)
```

4.2.34 getModelState

Gets the model state of the given model cref.

```
# not available
```

4.2.35 getReal

Get real value.

```
value, status = oms_getReal(cref)
```

4.2.36 getSolver

Gets the selected solver method of the given system.

```
solver, status = oms_getSolver(cref)
```

4.2.37 getStartTime

Get the start time from the model.

```
startTime, status = oms_getStartTime(cref)
```

4.2.38 getStopTime

Get the stop time from the model.

```
stopTime, status = oms_getStopTime(cref)
```

4.2.39 getSubModelPath

Returns the path of a given component.

```
# not available
```

4.2.40 getSystemType

Gets the type of the given system.

```
# not available
```

4.2.41 getTLMBus

Gets the TLM bus objects of the given TLM bus cref.

```
# not available
```

4.2.42 getTLMVariableTypes

Gets the type of an TLM variable.

```
# not available
```

4.2.43 getTolerance

Gets the tolerance of a given system or component.

```
absoluteTolerance, relativeTolerance, status = oms_getTolerance(cref)
```

4.2.44 getVariableStepSize

Gets the step size parameters.

```
initialStepSize, minimumStepSize, maximumStepSize, status = oms_
↪getVariableStepSize(cref)
```

4.2.45 getVersion

Returns the library's version string.

```
version = oms_getVersion()
```

4.2.46 importFile

Imports a composite model from a SSP file.

```
cref, status = oms_importFile(filename)
```

4.2.47 importString

Imports a composite model from a string.

```
cref, status = oms_importString(contents)
```

4.2.48 initialize

Initializes a composite model.

```
status = oms_initialize(cref)
```

4.2.49 instantiate

Instantiates a given composite model.

```
status = oms_instantiate(cref)
```

4.2.50 list

Lists the SSD representation of a given model, system, or component.

Memory is allocated for *contents*. The caller is responsible to free it using the C-API. The Lua and Python bindings take care of the memory and the caller doesn't need to call free.

```
contents, status = oms_list(cref)
```

4.2.51 listUnconnectedConnectors

Lists all unconnected connectors of a given system.

Memory is allocated for *contents*. The caller is responsible to free it using the C-API. The Lua and Python bindings take care of the memory and the caller doesn't need to call free.

```
contents, status = oms_listUnconnectedConnectors(cref)
```

4.2.52 newModel

Creates a new and yet empty composite model.

```
status = oms_newModel(cref)
```

4.2.53 parseModelName

Parses the model name from a given SSD representation.

Memory is allocated for *ident*. The caller is responsible to free it using the C-API. The Lua and Python bindings take care of the memory and the caller doesn't need to call free.

```
ident, status = oms_parseModelName(contents)
```

4.2.54 removeSignalsFromResults

Removes all variables that match the given regex to the result file.

```
status = oms_removeSignalsFromResults(cref, regex)
```

The second argument, i.e. regex, is considered as a regular expression (C++11). ".*" and "*(.)*" can be used to hit all variables.

4.2.55 rename

Renames a model, system, or component.

```
status = oms_rename(cref, newCref)
```

4.2.56 reset

Reset the composite model after a simulation run.

The FMUs go into the same state as after instantiation.

```
status = oms_reset(cref)
```

4.2.57 setActivationRatio

Experimental feature for setting the activation ratio of FMUs for experimenting with multi-rate master algorithms.

```
status = experimental_setActivationRatio(cref, k)
```

4.2.58 setBoolean

Sets the value of a given boolean signal.

```
status = oms_setBoolean(cref, value)
```

4.2.59 setBusGeometry

```
# not available
```

4.2.60 setCommandLineOption

Sets special flags.

```
status = oms_setCommandLineOption(cmd)
```

Available flags:

```
info:    Usage: OMSimulator [Options] [Lua script] [FMU] [SSP file]
         Options:
           --clearAllOptions                  Reset all flags to default␣
↪values
           --fetchAllVars=<arg>               Workaround for certain FMUs␣
↪that do not update all internal dependencies automatically
           --help [-h]                        Displays the help text
           --ignoreInitialUnknowns=<bool>     Ignore the initial unknowns␣
↪from the modelDescription.xml
```

(continues on next page)

(continued from previous page)

```
          --inputExtrapolation=<bool>     Enables input extrapolation␣
↪using derivative information
          --intervals=<int> [-i]          Specifies the number of␣
↪communication points (arg > 1)
          --logFile=<arg> [-l]            Specifies the logfile (stdout␣
↪is used if no log file is specified)
          --logLevel=<int>                0 default, 1 debug, 2␣
↪debug+trace
          --mode=<arg> [-m]               Forces a certain FMI mode iff␣
↪the FMU provides cs and me ([cs], me)
          --numProcs=<int> [-n]           Specifies the max. number of␣
↪processors to use (0=auto, 1=default)
          --progressBar=<bool>            Shows a progress bar for the␣
↪simulation progress in the terminal
          --realTime=<bool>               Experimental feature for (soft)␣
↪real-time co-simulation
          --resultFile=<arg> [-r]         Specifies the name of the␣
↪output result file
          --setInputDerivatives=<bool>    Deprecated; see '--
↪inputExtrapolation'
          --solver=<arg>                  Specifies the integration␣
↪method (euler, [cvode])
          --solverStats=<bool>            Adds solver stats to the result␣
↪file, e.g. step size (not supported for all solvers)
          --startTime=<double> [-s]       Specifies the start time
          --stopTime=<double> [-t]        Specifies the stop time
          --stripRoot=<bool>              Removes the root system prefix␣
↪from all exported signals
          --suppressPath=<bool>           Supresses path information in␣
↪info messages; especially useful for testing
          --tempDir=<arg>                 Specifies the temp directory
          --timeout=<int>                 Specifies the maximum allowed␣
↪time in seconds for running a simulation (0 disables)
          --tolerance=<double>            Specifies the relative tolerance
          --version [-v]                  Displays version information
          --wallTime=<bool>               Add wall time information for␣
↪to the result file
          --workingDir=<arg>              Specifies the working directory
```

4.2.61 setConnectionGeometry

```
# not available
```

4.2.62 setConnectorGeometry

Set geometry information to a given connector.

```
# not available
```

4.2.63 setElementGeometry

Set geometry information to a given component.

```
# not available
```

4.2.64 setFixedStepSize

Sets the fixed step size. Can be used for the communication step size of co-simulation systems and also for the integrator step size in model exchange systems.

```
status = oms_setFixedStepSize(cref, stepSize)
```

4.2.65 setInteger

Sets the value of a given integer signal.

```
status = oms_setInteger(cref, value)
```

4.2.66 setLogFile

Redirects logging output to file or std streams. The warning/error counters are reset.

filename="" to redirect to std streams and proper filename to redirect to file.

```
status = oms_setLogFile(filename)
```

4.2.67 setLoggingCallback

Sets a callback function for the logging system.

```
# not available
```

4.2.68 setLoggingInterval

Set the logging interval of the simulation.

```
status = oms_setLoggingInterval(cref, loggingInterval)
```

4.2.69 setLoggingLevel

Enables/Disables debug logging (logDebug and logTrace).

0 default, 1 default+debug, 2 default+debug+trace

```
oms_setLoggingLevel(logLevel)
```

4.2.70 setMaxLogFileSize

Sets maximum log file size in MB. If the file exceeds this limit, the logging will continue on stdout.

```
oms_setMaxLogFileSize(size)
```

4.2.71 setReal

Sets the value of a given real signal.

```
status = oms_setReal(cref, value)
```

This function can be called in different model states:

- Before instantiation: *setReal* can be used to set start values or to define initial unknowns (e.g. parameters, states). The values are not immediately applied to the simulation unit, since it isn't actually instantiated.
- After instantiation and before initialization: Same as before instantiation, but the values are applied immediately to the simulation unit.
- After initialization: Can be used to force external inputs, which might cause discrete changes of continuous signals.

4.2.72 setRealInputDerivative

Sets the first order derivative of a real input signal.

This can only be used for CS-FMU real input signals.

```
status = oms_setRealInputDerivative(cref, value)
```

4.2.73 setResultFile

Set the result file of the simulation.

```
status = oms_setResultFile(cref, filename)
status = oms_setResultFile(cref, filename, bufferSize)
```

The creation of a result file is omitted if the filename is an empty string.

4.2.74 setSignalFilter

```
status = oms_setSignalFilter(cref, regex)
```

4.2.75 setSolver

Sets the solver method for the given system.

```
status = oms_setSolver(cref, solver)
```

solver	Type	Description
oms_solver_sc_explicit_euler	sc-system	Explicit euler with fixed step size
oms_solver_sc_cvode	sc-system	CVODE with adaptive stepsize
oms_solver_wc_ma	wc-system	default master algorithm with fixed step size
oms_solver_wc_mav	wc-system	master algorithm with adaptive stepsize
oms_solver_wc_mav2	wc-system	master algorithm with adaptive stepsize (double-step)

4.2.76 setStartTime

Set the start time of the simulation.

```
status = oms_setStartTime(cref, startTime)
```

4.2.77 setStopTime

Set the stop time of the simulation.

```
status = oms_setStopTime(cref, stopTime)
```

4.2.78 setTLMBusGeometry

```
# not available
```

4.2.79 setTLMConnectionParameters

Simulates a composite model in its own thread.

```
# not available
```

4.2.80 setTLMPositionAndOrientation

Sets initial position and orientation for a TLM 3D interface.

```
status oms_setTLMPositionAndOrientation(cref, x1, x2, x3, A11, A12, A13,␣
↪A21, A22, A23, A31, A32, A33)
```

4.2.81 setTLMSocketData

Sets data for TLM socket communication.

```
status = oms_setTLMSocketData(cref, address, managerPort, monitorPort)
```

4.2.82 setTempDirectory

Set new temp directory.

```
status = oms_setTempDirectory(newTempDir)
```

4.2.83 setTolerance

Sets the tolerance for a given model or system.

```
status = oms_setTolerance(const char* cref, double tolerance)
status = oms_setTolerance(const char* cref, double absoluteTolerance,␣
↪double relativeTolerance)
```

Default values are *1e-4* for both relative and absolute tolerances.

A tolerance specified for a model is automatically applied to its root system, i.e. both calls do exactly the same:

```
oms_setTolerance("model", absoluteTolerance, relativeTolerance);
oms_setTolerance("model.root", absoluteTolerance, relativeTolerance);
```

Component, e.g. FMUs, pick up the tolerances from there system. That means it is not possible to define different tolerances for FMUs in the same system right now.

In a strongly coupled system (*oms_system_sc*), the relative tolerance is used for CVODE and the absolute tolerance is used to solve algebraic loops.

In a weakly coupled system (*oms_system_wc*), both the relative and absolute tolerances are used for the adaptive step master algorithms and the absolute tolerance is used to solve algebraic loops.

4.2.84 setVariableStepSize

Sets the step size parameters for methods with stepsize control.

```
status = oms_getVariableStepSize(cref, initialStepSize, minimumStepSize,␣
↪maximumStepSize)
```

4.2.85 setWorkingDirectory

Set a new working directory.

```
status = oms_setWorkingDirectory(newWorkingDir)
```

4.2.86 simulate

Simulates a composite model.

```
status = oms_simulate(cref)
```

4.2.87 simulate_asynchronous

Simulates a composite model in its own thread.

```
# not available
```

4.2.88 simulate_realtime

Experimental feature for (soft) real-time simulation.

```
status = experimental_simulate_realtime(ident)
```

4.2.89 stepUntil

Simulates a composite model until a given time value.

```
status = oms_stepUntil(cref, stopTime)
```

4.2.90 terminate

Terminates a given composite model.

```
status = oms_terminate(cref)
```

CHAPTER

FIVE

OMSIMULATORPYTHON

This is a shared library that provides a Python interface for the OMSimulatorLib library.

5.1 Examples

```python
from OMSimulator import OMSimulator
oms.setTempDirectory("./temp/")
oms.newModel("model")
oms.addSystem("model.root", oms.system_sc)

# instantiate FMUs
oms.addSubModel("model.root.system1", "FMUs/System1.fmu")
oms.addSubModel("model.root.system2", "FMUs/System2.fmu")

# add connections
oms.addConnection("model.root.system1.y", "model.root.system2.u")
oms.addConnection("model.root.system2.y", "model.root.system1.u")

# simulation settings
oms.setResultFile("model", "results.mat")
oms.setStopTime("model", 0.1)
oms.setFixedStepSize("model.root", 1e-4)

oms.instantiate("model")
oms.setReal("model.root.system1.x_start", 2.5)

oms.initialize("model")
oms.simulate("model")
oms.terminate("model")
oms.delete("model")
```

5.2 Python Scripting Commands

5.2.1 RunFile

Simulates a single FMU or SSP model.

```python
# not available
```

5.2.2 addBus

Adds a bus to a given component.

```
status = oms.addBus(cref)
```

5.2.3 addConnection

Adds a new connection between connectors *A* and *B*. The connectors need to be specified as fully qualified component references, e.g., *"model.system.component.signal"*.

```
status = oms.addConnection(crefA, crefB)
```

The two arguments *crefA* and *crefB* get swapped automatically if necessary.

5.2.4 addConnector

Adds a connector to a given component.

```
status = oms.addConnector(cref, causality, type)
```

5.2.5 addConnectorToBus

Adds a connector to a bus.

```
status = oms.addConnectorToBus(busCref, connectorCref)
```

5.2.6 addConnectorToTLMBus

Adds a connector to a TLM bus.

```
status = oms.addConnectorToTLMBus(busCref, connectorCref, type)
```

5.2.7 addExternalModel

Adds an external model to a TLM system.

```
status = oms.addExternalModel(cref, path, startscript)
```

5.2.8 addSignalsToResults

Add all variables that match the given regex to the result file.

```
status = oms.addSignalsToResults(cref, regex)
```

The second argument, i.e. regex, is considered as a regular expression (C++11). ".*" and *"(.)*"* can be used to hit all variables.

5.2.9 addSubModel

Adds a component to a system.

```
status = oms.addSubModel(cref, fmuPath)
```

5.2.10 addSystem

Adds a (sub-)system to a model or system.

```
status = oms.addSystem(cref, type)
```

5.2.11 addTLMBus

Adds a TLM bus.

```
status = oms.addTLMBus(cref, domain, dimensions, interpolation)
```

5.2.12 addTLMConnection

Connects two TLM connectors.

```
status = oms.addTLMConnection(crefA, crefB, delay, alpha, linearimpedance,␣
↪angularimpedance)
```

5.2.13 cancelSimulation_asynchronous

Cancels a running asynchronous simulation.

```
# not available
```

5.2.14 compareSimulationResults

This function compares a given signal of two result files within absolute and relative tolerances.

```
oms.compareSimulationResults(filenameA, filenameB, var, relTol, absTol)
```

5.2.15 copySystem

Copies a system.

```
status = oms.copySystem(source, target)
```

5.2.16 delete

Deletes a connector, component, system, or model object.

```
status = oms.delete(cref)
```

5.2.17 deleteConnection

Deletes the connection between connectors *crefA* and *crefB*.

```
status = oms.deleteConnection(crefA, crefB)
```

The two arguments *crefA* and *crefB* get swapped automatically if necessary.

5.2.18 deleteConnectorFromBus

Deletes a connector from a given bus.

```
status = oms.deleteConnectorFromBus(busCref, connectorCref)
```

5.2.19 deleteConnectorFromTLMBus

Deletes a connector from a given TLM bus.

```
status = oms.deleteConnectorFromTLMBus(busCref, connectorCref)
```

5.2.20 export

Exports a composite model to a SPP file.

```
status = oms.export(cref, filename)
```

5.2.21 exportDependencyGraphs

Export the dependency graphs of a given model to dot files.

```
status = oms.exportDependencyGraphs(cref, initialization, simulation)
```

5.2.22 extractFMIKind

Extracts the FMI kind of a given FMU from the file system.

```
# not available
```

5.2.23 freeMemory

Free the memory allocated by some other API. Pass the object for which memory is allocated.

```
oms.freeMemory(obj)
```

5.2.24 getBoolean

Get boolean value of given signal.

```
value, status = oms.getBoolean(cref)
```

5.2.25 getBus

Gets the bus object.

```
# not available
```

5.2.26 getComponentType

Gets the type of the given component.

```
# not available
```

5.2.27 getConnections

Get list of all connections from a given component.

```
# not available
```

5.2.28 getConnector

Gets the connector object of the given connector cref.

```
# not available
```

5.2.29 getElement

Get element information of a given component reference.

```
# not available
```

5.2.30 getElements

Get list of all sub-components of a given component reference.

```
# not available
```

5.2.31 getFMUInfo

Returns FMU specific information.

```
# not available
```

5.2.32 getFixedStepSize

Gets the fixed step size. Can be used for the communication step size of co-simulation systems and also for the integrator step size in model exchange systems.

```
stepSize, status = oms.getFixedStepSize(cref)
```

5.2.33 getInteger

Get integer value of given signal.

```
value, status = oms.getInteger(cref)
```

5.2.34 getModelState

Gets the model state of the given model cref.

```
# not available
```

5.2.35 getReal

Get real value.

```
value, status = oms.getReal(cref)
```

5.2.36 getSolver

Gets the selected solver method of the given system.

```
solver, status = oms.getSolver(cref)
```

5.2.37 getStartTime

Get the start time from the model.

```
startTime, status = oms.getStartTime(cref)
```

5.2.38 getStopTime

Get the stop time from the model.

```
stopTime, status = oms.getStopTime(cref)
```

5.2.39 getSubModelPath

Returns the path of a given component.

```
path, status = oms.getSubModelPath(cref)
```

5.2.40 getSystemType

Gets the type of the given system.

```
# not available
```

5.2.41 getTLMBus

Gets the TLM bus objects of the given TLM bus cref.

```
# not available
```

5.2.42 getTLMVariableTypes

Gets the type of an TLM variable.

```
# not available
```

5.2.43 getTolerance

Gets the tolerance of a given system or component.

```
absoluteTolerance, relativeTolerance, status = oms.getTolerance(cref)
```

5.2.44 getVariableStepSize

Gets the step size parameters.

```
initialStepSize, minimumStepSize, maximumStepSize, status = oms.
↪getVariableStepSize(cref)
```

5.2.45 getVersion

Returns the library's version string.

```
oms = OMSimulator()
oms.getVersion()
```

5.2.46 importFile

Imports a composite model from a SSP file.

```
cref, status = oms.importFile(filename)
```

5.2.47 importString

Imports a composite model from a string.

```
cref, status = oms.importString(contents)
```

5.2.48 initialize

Initializes a composite model.

```
status = oms.initialize(cref)
```

5.2.49 instantiate

Instantiates a given composite model.

```
status = oms.instantiate(cref)
```

5.2.50 list

Lists the SSD representation of a given model, system, or component.

Memory is allocated for *contents*. The caller is responsible to free it using the C-API. The Lua and Python bindings take care of the memory and the caller doesn't need to call free.

```
contents, status = oms.list(cref)
```

5.2.51 listUnconnectedConnectors

Lists all unconnected connectors of a given system.

Memory is allocated for *contents*. The caller is responsible to free it using the C-API. The Lua and Python bindings take care of the memory and the caller doesn't need to call free.

```
contents, status = oms.listUnconnectedConnectors(cref)
```

5.2.52 newModel

Creates a new and yet empty composite model.

```
status = oms.newModel(cref)
```

5.2.53 parseModelName

Parses the model name from a given SSD representation.

Memory is allocated for *ident*. The caller is responsible to free it using the C-API. The Lua and Python bindings take care of the memory and the caller doesn't need to call free.

```
ident, status = oms.parseModelName(contents)
```

5.2.54 removeSignalsFromResults

Removes all variables that match the given regex to the result file.

```
status = oms.removeSignalsFromResults(cref, regex)
```

The second argument, i.e. regex, is considered as a regular expression (C++11). *".*"* and *"(.)*"* can be used to hit all variables.

5.2.55 rename

Renames a model, system, or component.

```
status = oms.rename(cref, newCref)
```

5.2.56 reset

Reset the composite model after a simulation run.

The FMUs go into the same state as after instantiation.

```
status = oms.reset(cref)
```

5.2.57 setActivationRatio

Experimental feature for setting the activation ratio of FMUs for experimenting with multi-rate master algorithms.

```
# not yet available
```

5.2.58 setBoolean

Sets the value of a given boolean signal.

```
status = oms.setBoolean(cref, value)
```

5.2.59 setBusGeometry

```
# not available
```

5.2.60 setCommandLineOption

Sets special flags.

```
status = oms.setCommandLineOption(cmd)
```

Available flags:

```
info:    Usage: OMSimulator [Options] [Lua script] [FMU] [SSP file]
         Options:
           --clearAllOptions                 Reset all flags to default␣
↪values
           --fetchAllVars=<arg>              Workaround for certain FMUs␣
↪that do not update all internal dependencies automatically
           --help [-h]                       Displays the help text
           --ignoreInitialUnknowns=<bool>    Ignore the initial unknowns␣
↪from the modelDescription.xml
           --inputExtrapolation=<bool>       Enables input extrapolation␣
↪using derivative information
           --intervals=<int> [-i]            Specifies the number of␣
↪communication points (arg > 1)
           --logFile=<arg> [-l]              Specifies the logfile (stdout␣
↪is used if no log file is specified)
           --logLevel=<int>                  0 default, 1 debug, 2␣
↪debug+trace
           --mode=<arg> [-m]                 Forces a certain FMI mode iff␣
↪the FMU provides cs and me ([cs], me)
           --numProcs=<int> [-n]             Specifies the max. number of␣
↪processors to use (0=auto, 1=default)
           --progressBar=<bool>              Shows a progress bar for the␣
↪simulation progress in the terminal
           --realTime=<bool>                 Experimental feature for (soft)␣
↪real-time co-simulation
           --resultFile=<arg> [-r]           Specifies the name of the␣
↪output result file
```

(continues on next page)

(continued from previous page)

```
          --setInputDerivatives=<bool>    Deprecated; see '--
↪inputExtrapolation'
          --solver=<arg>                  Specifies the integration␣
↪method (euler, [cvode])
          --solverStats=<bool>            Adds solver stats to the result␣
↪file, e.g. step size (not supported for all solvers)
          --startTime=<double> [-s]       Specifies the start time
          --stopTime=<double> [-t]        Specifies the stop time
          --stripRoot=<bool>              Removes the root system prefix␣
↪from all exported signals
          --suppressPath=<bool>           Supresses path information in␣
↪info messages; especially useful for testing
          --tempDir=<arg>                 Specifies the temp directory
          --timeout=<int>                 Specifies the maximum allowed␣
↪time in seconds for running a simulation (0 disables)
          --tolerance=<double>            Specifies the relative tolerance
          --version [-v]                  Displays version information
          --wallTime=<bool>               Add wall time information for␣
↪to the result file
          --workingDir=<arg>              Specifies the working directory
```

5.2.61 setConnectionGeometry

```
# not available
```

5.2.62 setConnectorGeometry

Set geometry information to a given connector.

```
# not available
```

5.2.63 setElementGeometry

Set geometry information to a given component.

```
# not available
```

5.2.64 setFixedStepSize

Sets the fixed step size. Can be used for the communication step size of co-simulation systems and also for the integrator step size in model exchange systems.

```
status = oms.setFixedStepSize(cref, stepSize)
```

5.2.65 setInteger

Sets the value of a given integer signal.

```
status = oms.setInteger(cref, value)
```

5.2.66 setLogFile

Redirects logging output to file or std streams. The warning/error counters are reset.

filename=”” to redirect to std streams and proper filename to redirect to file.

```
status = oms.setLogFile(filename)
```

5.2.67 setLoggingCallback

Sets a callback function for the logging system.

```
# not available
```

5.2.68 setLoggingInterval

Set the logging interval of the simulation.

```
status = oms.setLoggingInterval(cref, loggingInterval)
```

5.2.69 setLoggingLevel

Enables/Disables debug logging (logDebug and logTrace).

0 default, 1 default+debug, 2 default+debug+trace

```
oms.setLoggingLevel(logLevel)
```

5.2.70 setMaxLogFileSize

Sets maximum log file size in MB. If the file exceeds this limit, the logging will continue on stdout.

```
oms.setMaxLogFileSize(size)
```

5.2.71 setReal

Sets the value of a given real signal.

```
status = oms.setReal(cref, value)
```

This function can be called in different model states:

- Before instantiation: *setReal* can be used to set start values or to define initial unknowns (e.g. parameters, states). The values are not immediately applied to the simulation unit, since it isn’t actually instantiated.

- After instantiation and before initialization: Same as before instantiation, but the values are applied immediately to the simulation unit.
- After initialization: Can be used to force external inputs, which might cause discrete changes of continuous signals.

5.2.72 setRealInputDerivative

Sets the first order derivative of a real input signal.

This can only be used for CS-FMU real input signals.

```
status = oms.setRealInputDerivative(cref, value)
```

5.2.73 setResultFile

Set the result file of the simulation.

```
status = oms.setResultFile(cref, filename)
status = oms.setResultFile(cref, filename, bufferSize)
```

The creation of a result file is omitted if the filename is an empty string.

5.2.74 setSignalFilter

```
status = oms.setSignalFilter(cref, regex)
```

5.2.75 setSolver

Sets the solver method for the given system.

```
status = oms.setSolver(cref, solver)
```

5.2.76 setStartTime

Set the start time of the simulation.

```
status = oms.setStartTime(cref, startTime)
```

5.2.77 setStopTime

Set the stop time of the simulation.

```
status = oms.setStopTime(cref, stopTime)
```

5.2.78 setTLMBusGeometry

```
# not available
```

5.2.79 setTLMConnectionParameters

Simulates a composite model in its own thread.

```
# not available
```

5.2.80 setTLMPositionAndOrientation

Sets initial position and orientation for a TLM 3D interface.

```
# not yet available
```

5.2.81 setTLMSocketData

Sets data for TLM socket communication.

```
# not yet available
```

5.2.82 setTempDirectory

Set new temp directory.

```
status = oms.setTempDirectory(newTempDir)
```

5.2.83 setTolerance

Sets the tolerance for a given model or system.

```
status = oms.setTolerance(const char* cref, double tolerance)
status = oms.setTolerance(const char* cref, double absoluteTolerance,␣
↪double relativeTolerance)
```

Default values are *1e-4* for both relative and absolute tolerances.

A tolerance specified for a model is automatically applied to its root system, i.e. both calls do exactly the same:

```
oms_setTolerance("model", absoluteTolerance, relativeTolerance);
oms_setTolerance("model.root", absoluteTolerance, relativeTolerance);
```

Component, e.g. FMUs, pick up the tolerances from there system. That means it is not possible to define different tolerances for FMUs in the same system right now.

In a strongly coupled system (*oms_system_sc*), the relative tolerance is used for CVODE and the absolute tolerance is used to solve algebraic loops.

In a weakly coupled system (*oms_system_wc*), both the relative and absolute tolerances are used for the adaptive step master algorithms and the absolute tolerance is used to solve algebraic loops.

5.2.84 setVariableStepSize

Sets the step size parameters for methods with stepsize control.

```
status = oms.getVariableStepSize(cref, initialStepSize, minimumStepSize,␣
↪maximumStepSize)
```

5.2.85 setWorkingDirectory

Set a new working directory.

```
status = oms.setWorkingDirectory(newWorkingDir)
```

5.2.86 simulate

Simulates a composite model.

```
status = oms.simulate(cref)
```

5.2.87 simulate_asynchronous

Simulates a composite model in its own thread.

```
# not available
```

5.2.88 simulate_realtime

Experimental feature for (soft) real-time simulation.

```
# not yet available
```

5.2.89 stepUntil

Simulates a composite model until a given time value.

```
status = oms.stepUntil(cref, stopTime)
```

5.2.90 terminate

Terminates a given composite model.

```
status = oms.terminate(cref)
```

CHAPTER

SIX

OMEDIT

OMSimulator has an optional dependency to OpenModelica in order to utilize the graphical modelling editor OMEdit. This feature requires to install the full OpenModelica tool suite, which includes OMSimulator. The independent stand-alone version doesn't provide any graphical modelling editor.

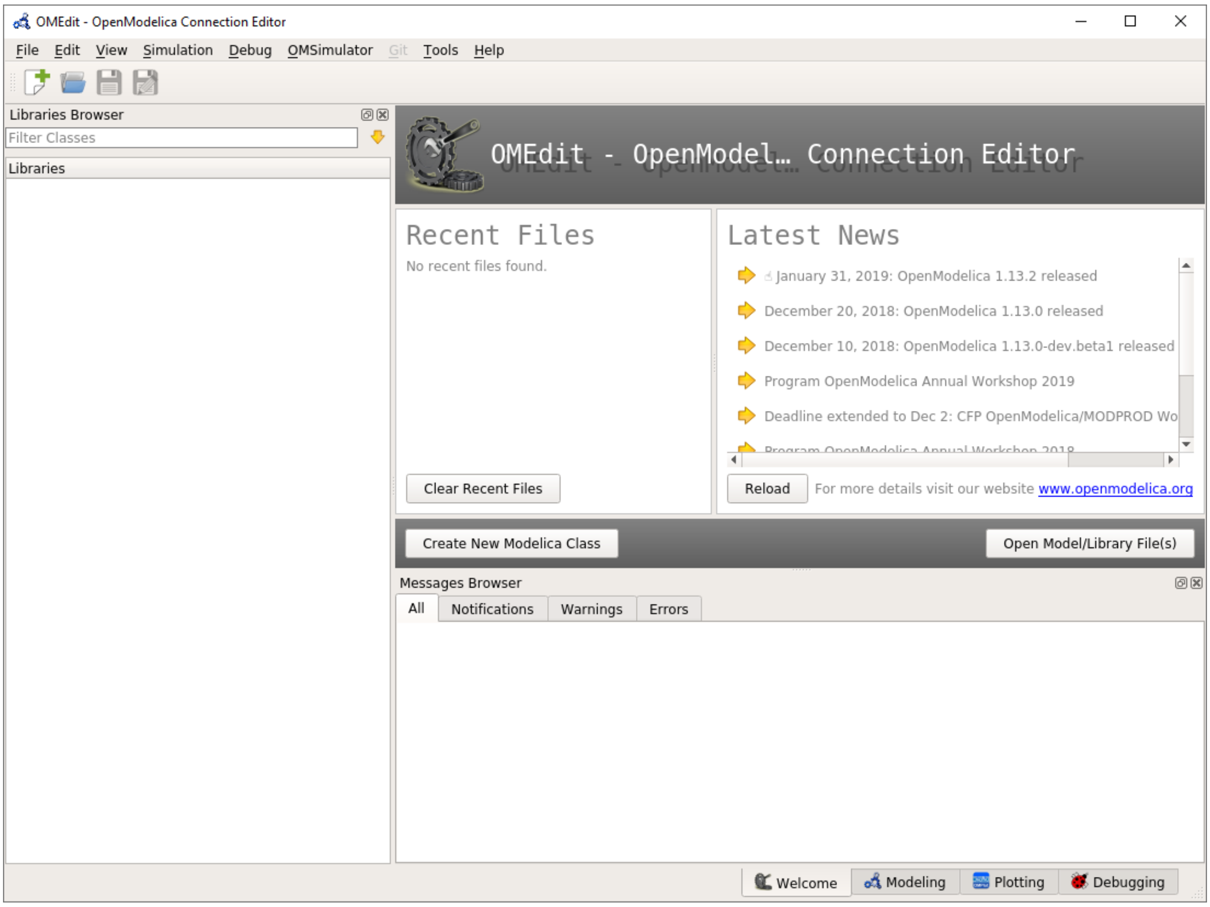

Fig. 1: OMEdit MainWindow and Browsers.

6.1 New OMSimulator Model

A new and empty OMSimulator model can be created from the OMSimulator menu item.

That will pop-up a dialog to enter the names of the model and the root system.

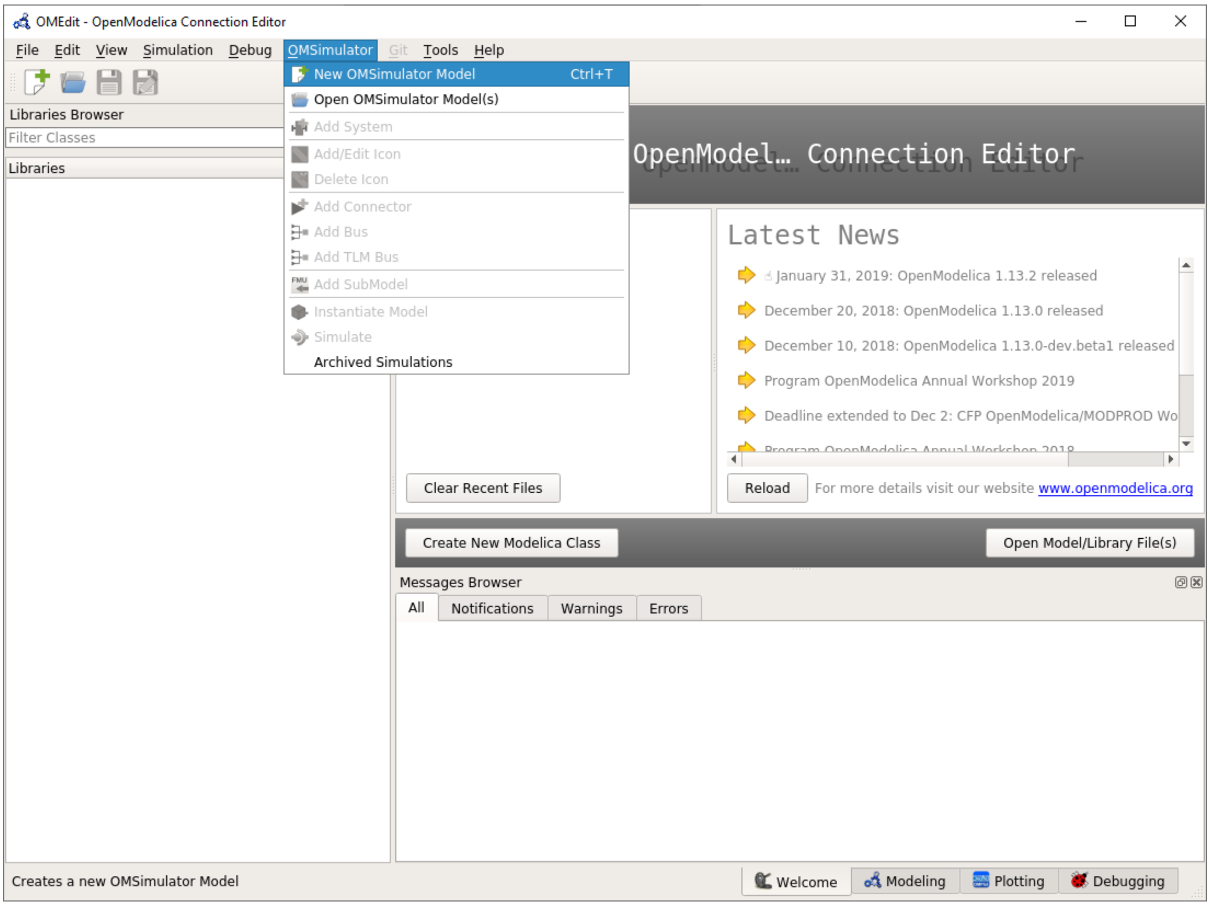

Fig. 2: OMEdit: New OMSimulator Model

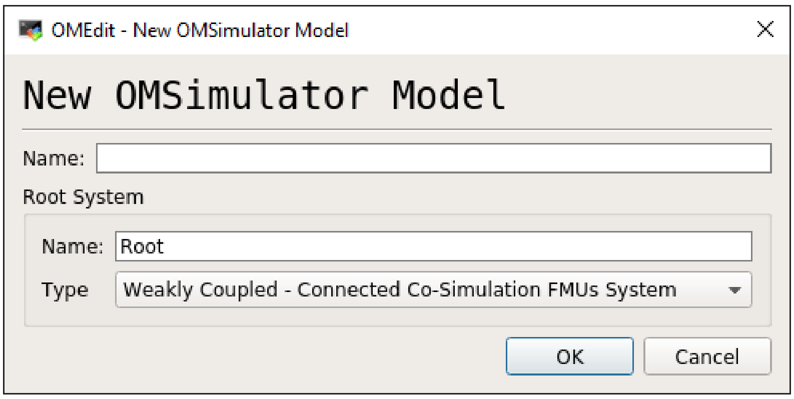

Fig. 3: OMEdit: New OMSimulator Model Dialog

6.2 Add System

A weakly coupled system (co-simulation) can integrate strongly coupled system (model exchange). Therefore, the weakly coupled system must to be selected from the Libraries Browser and the respective menu item can be selected:

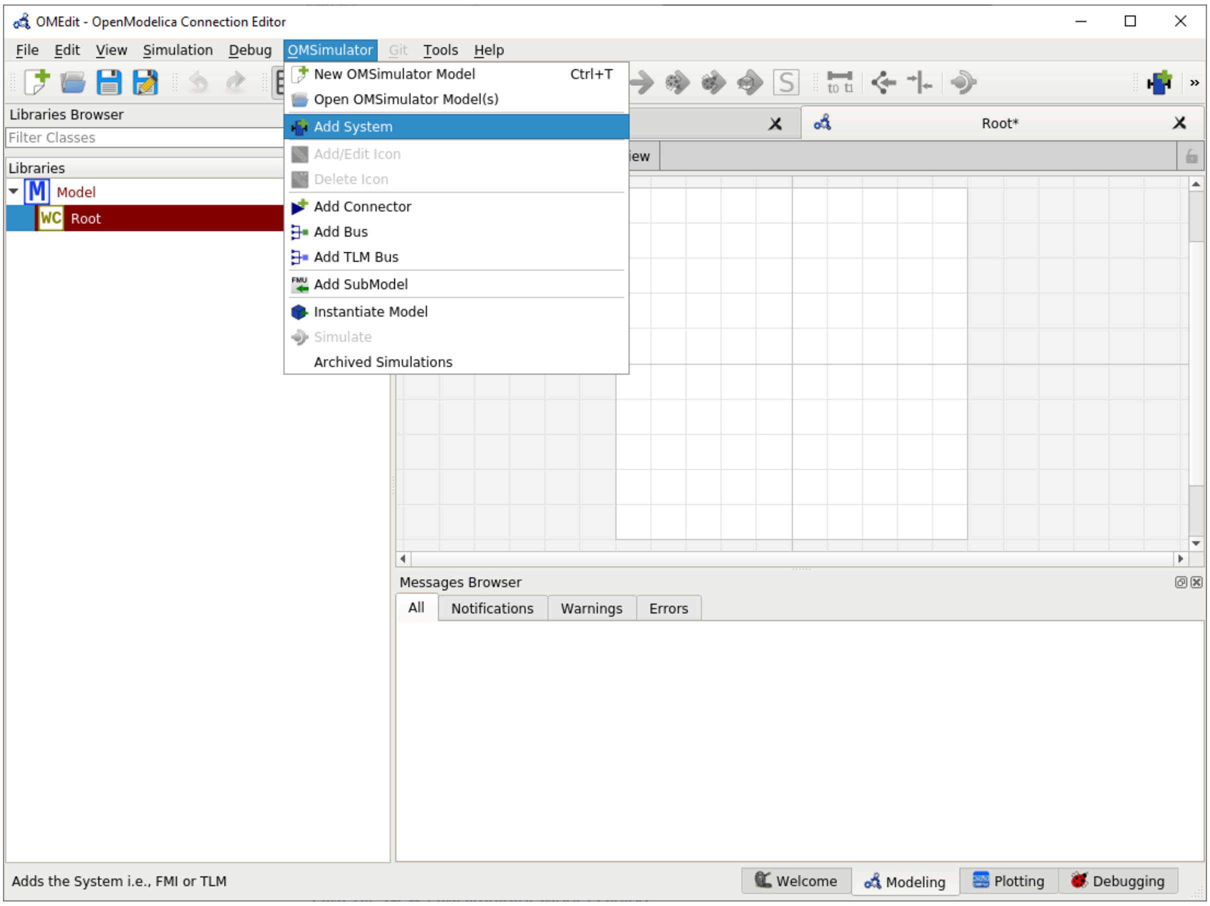

Fig. 4: OMEdit: Add System

That will pop-up a dialog to enter the names of the new system.

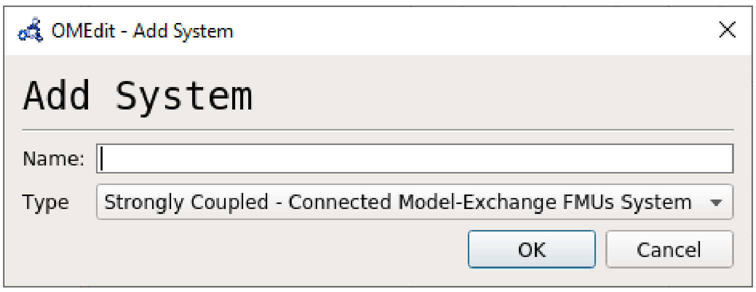

Fig. 5: OMEdit: Add System Dialog

6.3 Add SubModel

A sub-model is typically an FMU, but it also can be result file. In order to import a sub-model, the respective system must be selected and the action can be selected from the menu bar:

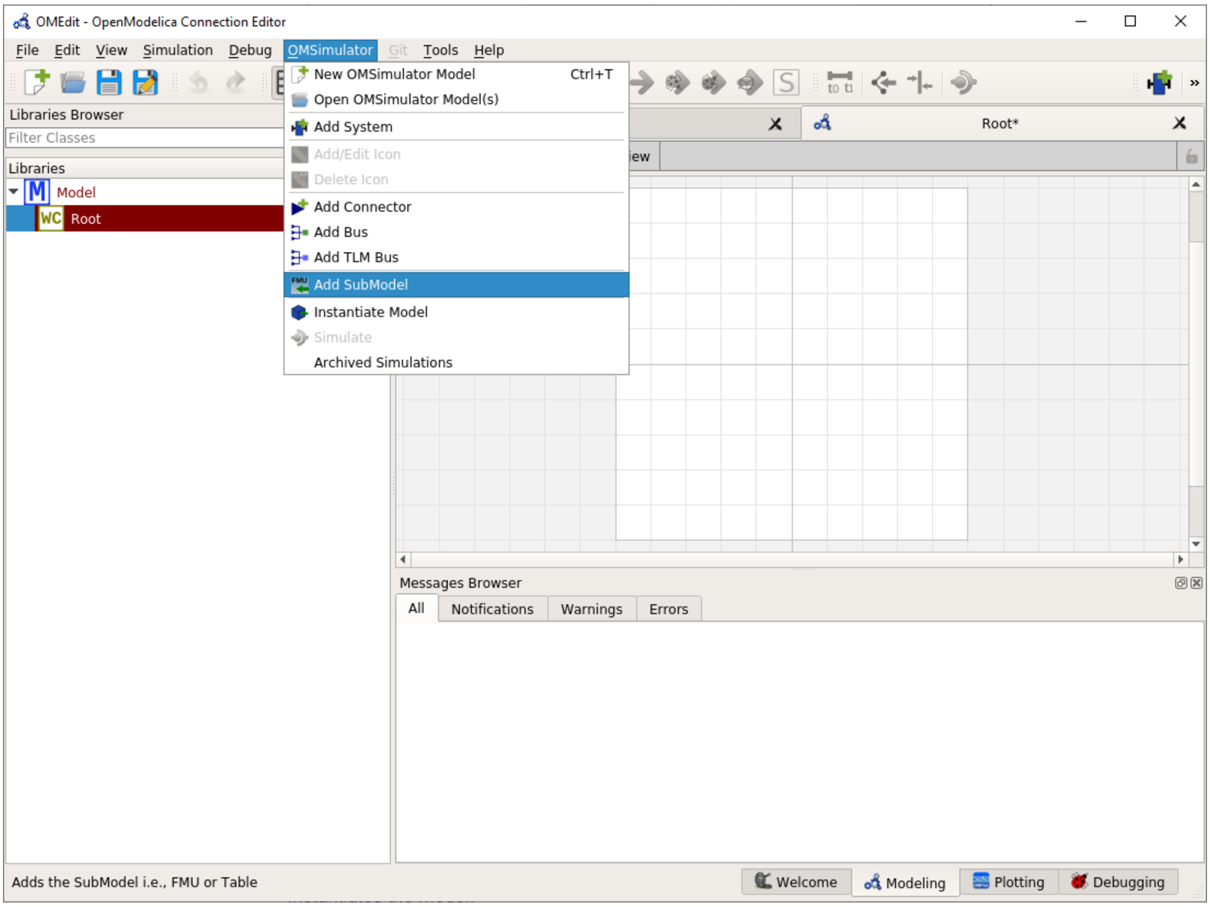

Fig. 6: OMEdit: Add SubModel

That will pop-up a dialog to enter the names of the new sub-model.

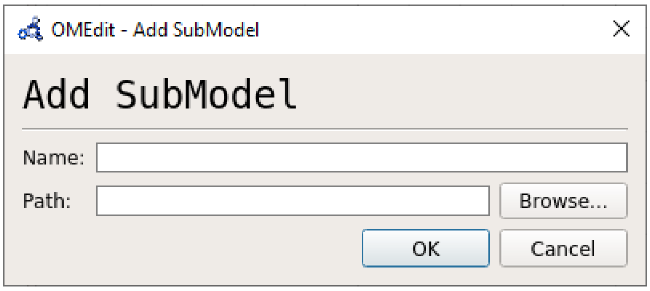

Fig. 7: OMEdit: Add SubModel Dialog

6.4 Instantiate Model

Before a OMSimulator model can be simulated, it needs to be instantiated. Therefore, select the dedicated menu item:

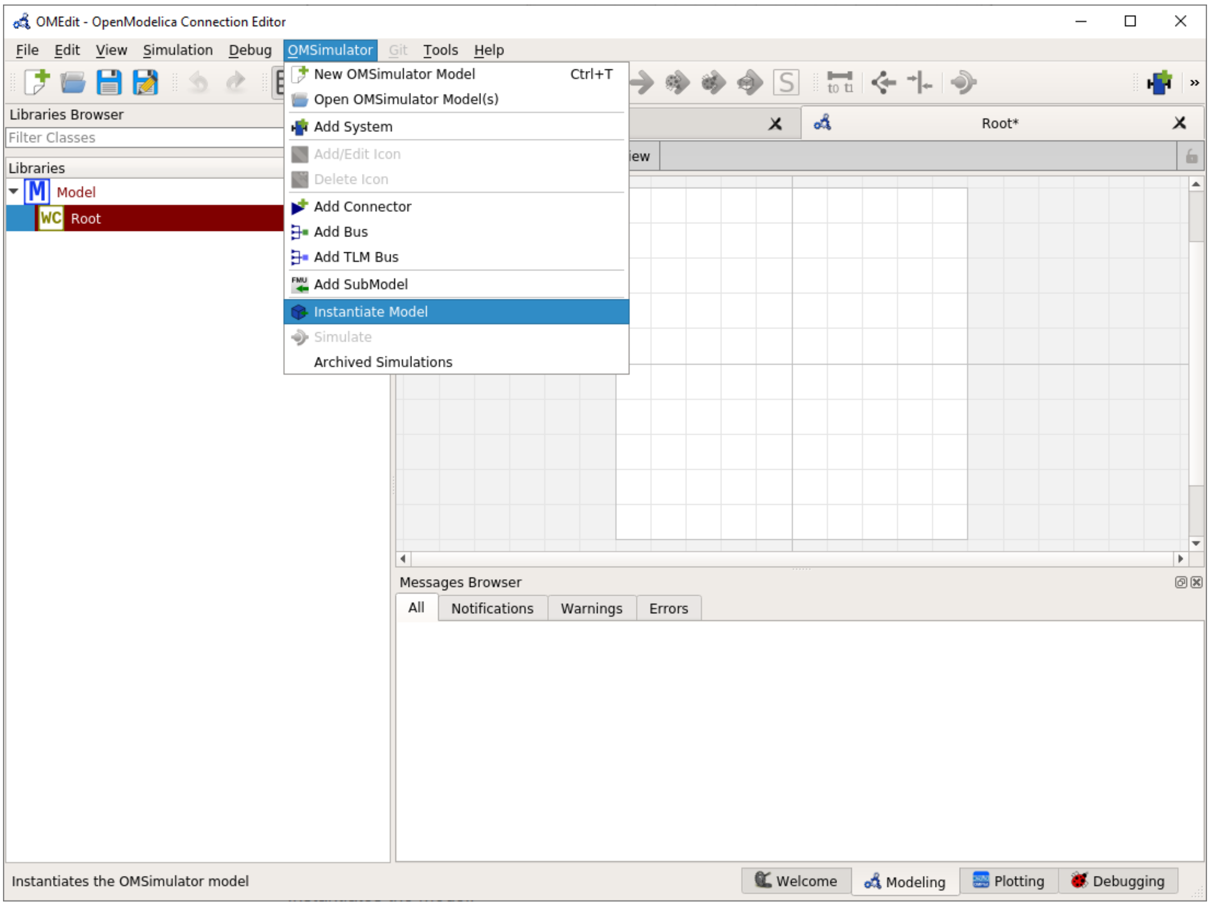

Fig. 8: OMEdit: Instantiate Model

That will pop-up a dialog to enter various simulation settings.

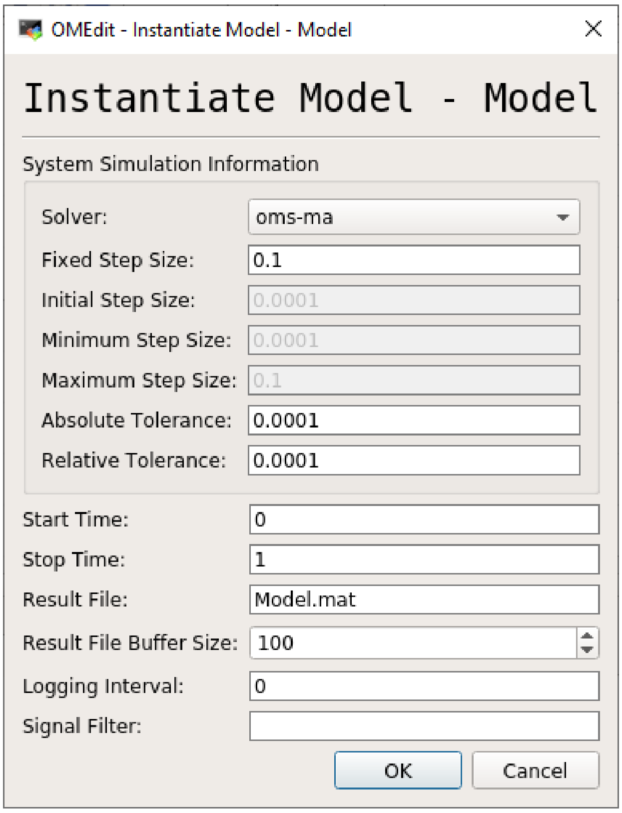

Fig. 9: OMEdit: Instantiate Model Dialog

6.5 Simulate

Select the simulate item from the OMSimulator menu.

CHAPTER

SEVEN

OMSYSIDENT

OMSysIdent is a module for the parameter estimation of behavioral models (wrapped as FMUs) on top of the OMSimulator API. It uses the Ceres solver (http://ceres-solver.org/) for the optimization task.

It is an optional module which can be disabled. Please check the build files for your platform for the respective flags.

7.1 Examples

There are examples in the testsuite below the subfolder *OMSysIdent* which use the scripting API. In addition there are examples which directly use the C API within the module's source code folder (*src/OMSysIdentLib*).

Below is a basic example from the testsuite (*HelloWorld_cs_Fit.lua*) which uses the Lua scripting API. It determines the parameters for the following "hello world" style Modelica model:

```
model HelloWorld
  parameter Real a = -1;
  parameter Real x_start = 1;
  Real x(start=x_start, fixed=true);
equation
  der(x) = a*x;
end HelloWorld;
```

The goal is to estimate the value of the coefficent *a* and the initial value *x_start* of the state variable *x*. Instead of real measurements, the script simply uses simulation data generated from the *HelloWorld* examples as measurement data. The array *data_time* contains the time instants at which a sample is taken and the array *data_x* contains the value of *x* that corresponds to the respective time instant.

The estimation parameters are defined by calls to function *omsi_addParameter(..)* in which the name of the parameter and a first guess for the parameter's value is stated.

Listing 1: HelloWorld_cs_Fit.lua

```
oms_setTempDirectory("./HelloWorld_cs_Fit/")
oms_newModel("HelloWorld_cs_Fit")
oms_addSystem("HelloWorld_cs_Fit.root", oms_system_wc)

-- add FMU
oms_addSubModel("HelloWorld_cs_Fit.root.HelloWorld", "../FMUs/HelloWorld.
↪fmu")
```

(continues on next page)

(continued from previous page)

```
-- create system identification model for model
simodel = omsi_newSysIdentModel("HelloWorld_cs_Fit");

-- Data generated from simulating HelloWorld.mo for 1.0s with Euler and 0.
↪1s step size
kNumSeries = 1;
kNumObservations = 11;
data_time = {0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1};
inputvars = {};
measurementvars = {"HelloWorld_cs_Fit.root.HelloWorld.x"};
data_x = {1, 0.9, 0.8100000000000001, 0.7290000000000001, 0.6561, 0.
↪5904900000000001, 0.5314410000000001, 0.4782969000000001, 0.43046721, 0.
↪387420489, 0.3486784401};

omsi_initialize(simodel, kNumSeries, data_time, inputvars, measurementvars)
-- omsi_describe(simodel)

omsi_addParameter(simodel, "HelloWorld_cs_Fit.root.HelloWorld.x_start", 0.
↪5);
omsi_addParameter(simodel, "HelloWorld_cs_Fit.root.HelloWorld.a", -0.5);
omsi_addMeasurement(simodel, 0, "HelloWorld_cs_Fit.root.HelloWorld.x",␣
↪data_x);
-- omsi_describe(simodel)

omsi_setOptions_max_num_iterations(simodel, 25)
omsi_solve(simodel, "BriefReport")

status, simodelstate = omsi_getState(simodel);
-- print(status, simodelstate)

status, startvalue1, estimatedvalue1 = omsi_getParameter(simodel,
↪"HelloWorld_cs_Fit.root.HelloWorld.a")
status, startvalue2, estimatedvalue2 = omsi_getParameter(simodel,
↪"HelloWorld_cs_Fit.root.HelloWorld.x_start")
-- print("HelloWorld.a startvalue=" .. startvalue1 .. ", estimatedvalue=" .
↪. estimatedvalue1)
-- print("HelloWorld.x_start startvalue=" .. startvalue2 .. ",␣
↪estimatedvalue=" .. estimatedvalue2)
is_OK1 = estimatedvalue1 > -1.1 and estimatedvalue1 < -0.9
is_OK2 = estimatedvalue2 > 0.9 and estimatedvalue2 < 1.1
print("HelloWorld.a estimation is OK: " .. tostring(is_OK1))
print("HelloWorld.x_start estimation is OK: " .. tostring(is_OK2))

omsi_freeSysIdentModel(simodel)

oms_terminate("HelloWorld_cs_Fit")
oms_delete("HelloWorld_cs_Fit")
```

Running the script generates the following console output:

```
iter      cost      cost_change  |gradient|   |step|    tr_ratio  tr_
↪radius  ls_iter  iter_time  total_time
 0  4.034320e-01    0.00e+00    2.19e+00   0.00e+00   0.00e+00  1.00e+04   ␣
↪      0     2.35e-02     2.35e-02
 1  3.821520e-02    3.65e-01    4.11e-01   9.87e-01   9.06e-01  2.15e+04   ␣
↪      1     2.90e-02     5.25e-02
```

(continues on next page)

(continued from previous page)

```
 2  6.837776e-04    3.75e-02    5.19e-02   3.58e-01   9.83e-01  6.46e+04  ␣
↪     1    2.77e-02    8.02e-02
 3  1.354499e-07    6.84e-04    6.08e-04   4.58e-02   1.00e+00  1.94e+05  ␣
↪     1    2.96e-02    1.10e-01
 4  5.854620e-15    1.35e-07    1.09e-07   7.22e-04   1.00e+00  5.82e+05  ␣
↪     1    3.08e-02    1.41e-01
 5  1.160287e-25    5.85e-15    2.26e-13   1.59e-07   1.00e+00  1.74e+06  ␣
↪     1    2.86e-02    1.69e-01
Ceres Solver Report: Iterations: 6, Initial cost: 4.034320e-01, Final␣
↪cost: 1.160287e-25, Termination: CONVERGENCE

=====================================
Total duration for parameter estimation: 169msec.
Result of parameter estimation (check 'Termination' status above whether␣
↪solver converged):

HelloWorld_cs_Fit.HelloWorld:a(start=-0.5, *estimate*=-1.04807)
HelloWorld_cs_Fit.HelloWorld:x_start(start=0.5, *estimate*=1)

=====================================
0      convergence
HelloWorld.a startvalue=-0.5, estimatedvalue=-1.0480741793778
HelloWorld.x_start startvalue=0.5, estimatedvalue=0.99999999999972
HelloWorld.a estimation is OK: true
HelloWorld.x_start estimation is OK: true
info:    Logging information has been saved to "HelloWorld_cs_Fit.log"
```

7.2 Lua Scripting Commands

7.2.1 omsi_addInput

Add input values for external model inputs.

If there are several measurement series, all series need to be conducted with the same external inputs!

```
-- simodel [inout] SysIdent model as opaque pointer.
-- var      [in] Name of input variable.
-- time     [in] Array of input time instants.
-- values   [in] Array of input values corresponding to respective "time"␣
↪array entries in omsi_initialize().
-- status   [out] Error status.
status = omsi_addInput(simodel, var, time, values)
```

7.2.2 omsi_addMeasurement

Add measurement values for a fitting variable.

```
-- simodel [inout] SysIdent model as opaque pointer.
-- iSeries [in] Index of measurement series.
-- var      [in] Name of variable.
-- values   [in] Array of measured values for respective time instants.
```

(continues on next page)

(continued from previous page)

```
-- status   [out] Error status.
status = omsi_addMeasurement(simodel, iSeries, var, values)
```

7.2.3 omsi_addParameter

Add parameter that should be estimated.

```
-- simodel    [inout] SysIdent model as opaque pointer.
-- var        [in] Name of parameter.
-- startvalue [in] Start value of parameter.
-- status     [out] Error status.
status = omsi_addParameter(simodel, var, startvalue)
```

7.2.4 omsi_describe

Print summary of SysIdent model.

```
-- simodel [inout] SysIdent model as opaque pointer.
-- status  [out] Error status.
status = omsi_describe(simodel)
```

7.2.5 omsi_freeSysIdentModel

Unloads a model.

```
-- simodel [inout] SysIdent model as opaque pointer.
omsi_freeSysIdentModel(simodel)
```

7.2.6 omsi_getParameter

Get parameter that should be estimated.

```
-- simodel        [inout] SysIdent model as opaque pointer.
-- var            [in] Name of parameter.
-- startvalue     [out] Start value of parameter.
-- estimatedvalue [out] Estimated value of parameter.
-- status         [out] Error status.
status, startvalue, estimatedvalue = omsi_getParameter(simodel, var)
```

7.2.7 omsi_getState

Get state of SysIdent model object.

```
-- simodel [inout] SysIdent model as opaque pointer.
-- state   [out] State of SysIdent model.
-- status  [out] Error status.
status, state = omsi_getState(simodel)
```

7.2.8 omsi_initialize

This function initializes a given composite model. After this call, the model is in simulation mode.

```
-- simodel         [inout] SysIdent model as opaque pointer.
-- nSeries         [in] Number of measurement series.
-- time            [in] Array of measurement/input time instants.
-- inputvars       [in] Array of names of input variables (empty array if␣
↪none).
-- measurmentvars [in] Array of names of observed measurement variables.
-- status          [out] Error status.
status = omsi_initialize(simodel, nSeries, time, inputvars,␣
↪measurementvars)
```

7.2.9 omsi_newSysIdentModel

Creates an empty model for parameter estimation.

```
-- ident    [in]  Name of the model instance as string.
-- simodel [out] SysIdent model instance as opaque pointer.
simodel = omsi_newSysIdentModel(ident)
```

7.2.10 omsi_setOptions_max_num_iterations

Set Ceres solver option *Solver::Options::max_num_iterations*.

```
-- simodel             [inout] SysIdent model as opaque pointer.
-- max_num_iterations [in] Maximum number of iterations for which the␣
↪solver
--                          should run (default: 25).
-- status              [out] Error status.
status = omsi_setOptions_max_num_iterations(simodel, max_num_iterations)
```

7.2.11 omsi_solve

Solve parameter estimation problem.

```
-- simodel     [inout] SysIdent model as opaque pointer.
-- reporttype [in] Print report and progress information after call to␣
↪Ceres solver.
--                 Supported report types: {"", "BriefReport", "FullReport
↪"},
--                 where "" denotes no output.
-- status      [out] Error status.
status = omsi_solve(simodel, reporttype)
```

CHAPTER

EIGHT

SSP SUPPORT

Composite models are imported and exported in the *System Structure Description (SSD)* format, which is part of the *System Structure and Parameterization (SSP)* standard.

8.1 Bus Connections

Bus connections are saved as annotations to the SSD file. Bus connectors are only allowed in weakly coupled and strongly coupled systems. Bus connections can exist in any system type. Bus connectors are used to hide SSD connectors and bus connections are used to hide existing SSD connections in the graphical user interface. It is not required that all connectors referenced in a bus are connected. One bus may be connected to multiple other buses, and also to SSD connectors.

The example below contains a root system with two subsystems, `WC1` and `WC2`. Bus connector `WC1.bus1` is connected to `WC2.bus2`. Bus connector `WC2.bus2` is also connected to SSD connector `WC1.C3`.

```
<?xml version="1.0" encoding="UTF-8"?>
<ssd:SystemStructureDescription name="Test" version="Draft20180219">
  <ssd:System name="Root">
    <ssd:Elements>
      <ssd:System name="WC2">
        <ssd:Connectors>
          <ssd:Connector name="C1" kind="input" type="Real"/>
          <ssd:Connector name="C2" kind="output" type="Real"/>
        </ssd:Connectors>
        <ssd:Annotations>
          <ssc:Annotation type="org.openmodelica">
            <oms:Bus name="bus2">
              <oms:Signals>
                <oms:Signal name="C1"/>
                <oms:Signal name="C2"/>
              </oms:Signals>
            </oms:Bus>
          </ssc:Annotation>
        </ssd:Annotations>
      </ssd:System>
      <ssd:System name="WC1">
        <ssd:Connectors>
          <ssd:Connector name="C1" kind="output" type="Real"/>
          <ssd:Connector name="C2" kind="input" type="Real"/>
          <ssd:Connector name="C3" kind="input" type="Real"/>
        </ssd:Connectors>
```

(continues on next page)

(continued from previous page)

```
        <ssd:Annotations>
          <ssc:Annotation type="org.openmodelica">
            <oms:Bus name="bus1">
              <oms:Signals>
                <oms:Signal name="C1"/>
                <oms:Signal name="C2"/>
              </oms:Signals>
            </oms:Bus>
          </ssc:Annotation>
        </ssd:Annotations>
      </ssd:System>
    </ssd:Elements>
    <ssd:Connections>
      <ssd:Connection startElement="WC2" startConnector="C1"
                      endElement="WC1" endConnector="C1"/>
      <ssd:Connection startElement="WC2" startConnector="C2"
                      endElement="WC1" endConnector="C2"/>
      <ssd:Connection startElement="WC2" startConnector="C2"
                      endElement="WC1" endConnector="C3"/>
    </ssd:Connections>
    <ssd:Annotations>
      <ssc:Annotation type="org.openmodelica">
        <oms:Connections>
          <oms:Connection startElement="WC1" startConnector="bus1"
                          endElement="WC2" endConnector="bus2"/>
          <oms:Connection startElement="WC2" startConnector="bus2"
                          endElement="WC1" endConnector="C3"/>
        </oms:Connections>
      </ssc:Annotation>
    </ssd:Annotations>
  </ssd:System>
</ssd:SystemStructureDescription>
```

8.2 TLM Systems

TLM systems are only allowed on top-level. SSD annotations are used to specify the system type inside the `ssd:SimulationInformation` tag, as shown in the example below. Attributes `ip`, `managerport` and `monitorport` defines the socket communication, used both to exchange data with external tools and with internal simulation threads.

```
<?xml version="1.0"?>
<ssd:System name="tlm">
  <ssd:SimulationInformation>
   <ssd:Annotations>
      <ssd:Annotation type="org.openmodelica">
        <oms:TlmMaster ip="127.0.1.1" managerport="11111" monitorport=
↪"11121"/>
      </ssd:Annotation>
   </ssd:Annotations>
  </ssd:SimulationInformation>
  <ssd:Elements>
    <ssd:System name="weaklycoupled">
      <ssd:SimulationInformation>
```

(continues on next page)

(continued from previous page)

```
        <ssd:FixedStepMaster description="oms-ma" stepSize="1e-1" />
      </ssd:SimulationInformation>
    </ssd:System>
  </ssd:Elements>
</ssd:System>
```

8.3 TLM Connections

TLM connections are implemented without regular SSD connections. TLM connections are only allowed in TLM systems. TLM connectors are only allowed in weakly coupled or strongly coupled systems. Both connectors and connections are implemented as SSD annotations in the System tag.

The example below shows a TLM system containing two weakly coupled systems, `wc1` and `wc2`. System `wc1` contains two TLM connectors, one of type 1D signal and one of type 1D mechanical. System `wc2` contains only a 1D signal type connector. The two 1D signal connectors are connected to each other in the TLM top-level system.

```
<?xml version="1.0"?>
<ssd:System name="tlm">
  <ssd:Elements>
    <ssd:System name="wc2">
      <ssd:Connectors>
        <ssd:Connector name="y" kind="input" type="Real" />
      </ssd:Connectors>
      <ssd:Annotations>
        <ssd:Annotation type="org.openmodelica">
          <oms:Bus name="bus2" type="tlm" domain="signal"
                   dimension="1" interpolation="none">
            <oms:Signals>
              <oms:Signal name="y" tlmType="value" />
            </oms:Signals>
          </oms:Bus>
        </ssd:Annotation>
      </ssd:Annotations>
    </ssd:System>
    <ssd:System name="wc1">
      <ssd:Connectors>
        <ssd:Connector name="y" kind="output" type="Real" />
        <ssd:Connector name="x" kind="output" type="Real" />
        <ssd:Connector name="v" kind="output" type="Real" />
        <ssd:Connector name="f" kind="input" type="Real" />
      </ssd:Connectors>
      <ssd:Annotations>
        <ssd:Annotation type="org.openmodelica">
          <oms:Bus name="bus1" type="tlm" domain="signal"
                   dimension="1" interpolation="none">
            <oms:Signals>
              <oms:Signal name="y" tlmType="value" />
            </oms:Signals>
          </oms:Bus>
          <oms:Bus name="bus2" type="tlm" domain="mechanical"
                   dimension="1" interpolation="none">
            <oms:Signals>
```

(continues on next page)

(continued from previous page)

```
                <oms:Signal name="x" tlmType="state" />
                <oms:Signal name="v" tlmType="flow" />
                <oms:Signal name="f" tlmType="effort" />
              </oms:Signals>
            </oms:Bus>
          </ssd:Annotation>
        </ssd:Annotations>
      </ssd:System>
    </ssd:Elements>
    <ssd:Annotations>
      <ssd:Annotation type="org.openmodelica">
        <oms:Connections>
          <oms:Connection startElement="wc1" startConnector="bus1"
                          endElement="wc2" endConnector="bus2"
                          delay="0.001000" alpha="0.300000"
                          linearimpedance="100.000000"
                          angularimpedance="0.000000" />
        </oms:Connections>
      </ssd:Annotation>
    </ssd:Annotations>
  </ssd:System>
```

Depending on the type of TLM bus connector (dimension, domain and interpolation), connectors need to be assigned to different tlm variable types. Below is the complete list of supported TLM bus types and their respective connectors.

1D signal

tlmType	causality
"value"	input/output

1D physical (no interpolation)

tlmType	causality
"state"	output
"flow"	output
"effort"	input

1D physical (coarse-grained interpolation)

tlmType	causality
"state"	output
"flow"	output
"wave"	input
"impedance"	input

1D physical (fine-grained interpolation)

tlmType	causality
`"state"`	output
`"flow"`	output
`"wave1"`	input
`"wave2"`	input
`"wave3"`	input
`"wave4"`	input
`"wave5"`	input
`"wave6"`	input
`"wave7"`	input
`"wave8"`	input
`"wave9"`	input
`"wave10"`	input
`"time1"`	input
`"time2"`	input
`"time3"`	input
`"time4"`	input
`"time5"`	input
`"time6"`	input
`"time7"`	input
`"time8"`	input
`"time9"`	input
`"time10"`	input
`"impedance"`	input

3D physical (no interpolation)

tlmType	causality
"state1"	output
"state2"	output
"state3"	output
"A11"	output
"A12"	output
"A13"	output
"A21"	output
"A22"	output
"A23"	output
"A31"	output
"A32"	output
"A33"	output
"flow1"	output
"flow2"	output
"flow3"	output
"flow4"	output
"flow5"	output
"flow6"	output
"effort1"	input
"effort2"	input
"effort3"	input
"effort4"	input
"effort5"	input
"effort6"	input

3D physical (coarse-grained interpolation)

tlmType	causality
"state1"	output
"state2"	output
"state3"	output
"A11"	output
"A12"	output
"A13"	output
"A21"	output
"A22"	output
"A23"	output
"A31"	output
"A32"	output
"A33"	output
"flow1"	output
"flow2"	output
"flow3"	output
"flow4"	output
"flow5"	output
"flow6"	output
"wave1"	input
"wave2"	input
"wave3"	input
"wave4"	input
"wave5"	input
"wave6"	input
"linearimpedance"	input
"angularimpedance"	input

3D physical (fine-grained interpolation)

tlmType	causality
"state1"	output
"state2"	output
"state3"	output
"A11"	output
"A12"	output
"A13"	output
"A21"	output
"A22"	output
"A23"	output
"A31"	output
"A32"	output
"A33"	output
"flow1"	output
"flow2"	output
"flow3"	output
"flow4"	output
"flow5"	output

Continued on next page

Table 1 – continued from previous page

tlmType	causality
"flow6"	output
"wave1_1"	input
"wave1_2"	input
"wave1_3"	input
"wave1_4"	input
"wave1_5"	input
"wave1_6"	input
"wave2_1"	input
"wave2_2"	input
"wave2_3"	input
"wave2_4"	input
"wave2_5"	input
"wave2_6"	input
"wave3_1"	input
"wave3_2"	input
"wave3_3"	input
"wave3_4"	input
"wave3_5"	input
"wave3_6"	input
"wave4_1"	input
"wave4_2"	input
"wave4_3"	input
"wave4_4"	input
"wave4_5"	input
"wave4_6"	input
"wave5_1"	input
"wave5_2"	input
"wave5_3"	input
"wave5_4"	input
"wave5_5"	input
"wave5_6"	input
"wave6_1"	input
"wave6_2"	input
"wave6_3"	input
"wave6_4"	input
"wave6_5"	input
"wave6_6"	input
"wave7_1"	input
"wave7_2"	input
"wave7_3"	input
"wave7_4"	input
"wave7_5"	input
"wave7_6"	input
"wave8_1"	input
"wave8_2"	input
"wave8_3"	input
"wave8_4"	input

Continued on next page

Table 1 – continued from previous page

tlmType	causality
"wave8_5"	input
"wave8_6"	input
"wave9_1"	input
"wave9_2"	input
"wave9_3"	input
"wave9_4"	input
"wave9_5"	input
"wave9_6"	input
"wave10_1"	input
"wave10_2"	input
"wave10_3"	input
"wave10_4"	input
"wave10_5"	input
"wave10_6"	input
"time1"	input
"time2"	input
"time3"	input
"time4"	input
"time5"	input
"time6"	input
"time7"	input
"time8"	input
"time9"	input
"time10"	input
"linearimpedance"	input
"angularimpedance"	input

INDEX