# CONTENTS

1 Introduction  

2 OMSimulator  
   2.1 OMSimulator Flags  
   2.2 Examples  

3 OMSimulatorLib  
   3.1 C-API  

4 OMSimulatorLua  
   4.1 Examples  
   4.2 Lua Scripting Commands  

5 OMSimulatorPython  
   5.1 Examples  
   5.2 Python Scripting Commands  

6 OpenModelicaScripting  
   6.1 Examples  
   6.2 OpenModelica Scripting Commands  

7 Graphical Modelling  
   7.1 New OMSimulator Model  
   7.2 Add System  
   7.3 Add SubModel  
   7.4 Instantiate Model  
   7.5 Simulate  

8 SSP Support  
   8.1 Bus Connections  
   8.2 TLM Systems  
   8.3 TLM Connections  

Index
The OMSimulator project is a FMI-based co-simulation tool that supports ordinary (i.e., non-delayed) and TLM connections. It supports large-scale simulation and virtual prototyping using models from multiple sources utilizing the FMI standard. It is integrated into OpenModelica but also available stand-alone, i.e., without dependencies to Modelica specific models or technology. OMSimulator provides an industrial-strength open-source FMI-based modelling and simulation tool. Input/output ports of FMUs can be connected, ports can be grouped to buses, FMUs can be parameterized and composed, and composite models can be exported according to the (preliminary) SSP (System Structure and Parameterization) standard. Efficient FMI based simulation is provided for both model-exchange and co-simulation. TLM-based tool connection is provided for a range of applications, e.g., Adams, Simulink, Beast, Dymola, and OpenModelica. Moreover, optional TLM (Transmission Line Modelling) domain-specific connectors are also supported, providing additional numerical stability to co-simulation. An external API is available for use from other tools and scripting languages such as Python and Lua.
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:
  --addParametersToCSV=<arg> Export parameters to .csv file,
  (true, [false])
  --clearAllOptions Reset all flags to default,
  --deleteTempFiles=<bool> Deletes temp files as soon as
  (true, false)
  --emitEvents=<bool> Specifies whether events should
  (true, false)
  --exportParametersInline=<arg> Export ParameterBindings inline
  with .ssd file,
  --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
  (true, false)
  --inputExtrapolation=<bool> Enables input extrapolation
  --intervals=<int> [-i] Specifies the number of
  communication points (arg > 1)
  --logFile=<arg> [-l] Specifies the logfile (stdout
  if no log file is specified)
  --logLevel=<int> 0 default, 1 debug, 2
  --maxEventIteration=<int> Specifies the max. number of
  iterations for handling a single event
  --maxLoopIteration=<int> Specifies the max. number of
  iterations for solving algebraic loops between system-level components.
  --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 (true, [false])
```

(continues on next page)
To use flag `logLevel` with option `debug` (`--logLevel=1`) or `debug+trace` (`--logLevel=2`) one needs to build OMSimulator with debug configuration enabled. Refer to the OMSimulator README on GitHub for further instructions.

### 2.2 Examples

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

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

#### 3.1.2 addBus

Adds a bus to a given component.

```c
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”.

```c
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.

```c
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.

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

3.1.6 addConnectorToTLMBus

Adds a connector to a TLM bus.

```c
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.

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

3.1.8 addSignalsToResults

[deprecated: `setSignalFilter` is the recommended API]

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

```c
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.

```c
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.

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

Adds a TLM bus.

```c
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.

```c
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.

```c
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.

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

The following table describes the input values:

<table>
<thead>
<tr>
<th>Input</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filenameA</td>
<td>String</td>
<td>Name of first result file to compare.</td>
</tr>
<tr>
<td>filenameB</td>
<td>String</td>
<td>Name of second result file to compare.</td>
</tr>
<tr>
<td>var</td>
<td>String</td>
<td>Name of signal to compare.</td>
</tr>
<tr>
<td>relTol</td>
<td>Number</td>
<td>Relative tolerance.</td>
</tr>
<tr>
<td>absTol</td>
<td>Number</td>
<td>Absolute tolerance.</td>
</tr>
</tbody>
</table>

The following table describes the return values:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>1 if the signal is considered as equal, 0 otherwise</td>
</tr>
</tbody>
</table>

3.1.15 copySystem

Copies a system.

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

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

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

3.1.17 deleteConnection

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

```c
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.

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

3.1.19 deleteConnectorFromTLMBus

Deletes a connector from a given TLM bus.

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

3.1.20 export

Exports a composite model to a SPP file.

```c
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.

```c
oms_status_enu_t oms_exportDependencyGraphs(const char* cref, const char* initialization, const char* event, const char* simulation);
```

3.1.22 exportSSVTemplate

Exports all signals that have start values of one or multiple FMUs to a SSV file that are read from `modelDescription.xml`. The function can be called for a top level model or a certain FMU component. If called for a top level model, start values of all FMUs are exported to the SSV file. If called for a component, start values of just this FMU are exported to the SSV file.
3.1.23 exportSnapshot

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.

```c
oms_status_enum_t oms_exportSnapshot(const char* cref, char** contents);
```

3.1.24 extractFMIKind

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

```c
oms_status_enum_t oms_extractFMIKind(const char* filename, oms_fmi_kind_enum_t* kind);
```

3.1.25 faultInjection

Defines a new fault injection block.

```c
oms_status_enum_t oms_faultInjection(const char* signal, oms_fault_type_enum_t faultType, double faultValue);
```

<table>
<thead>
<tr>
<th>type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oms_fault_type_bias</td>
<td>( y = y_{\text{original}} + \text{faultValue} )</td>
</tr>
<tr>
<td>oms_fault_type_gain</td>
<td>( y = y_{\text{original}} \times \text{faultValue} )</td>
</tr>
<tr>
<td>oms_fault_type_const</td>
<td>( y = \text{faultValue} )</td>
</tr>
</tbody>
</table>

3.1.26 freeMemory

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

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

3.1.27 getBoolean

Get boolean value of given signal.

```c
oms_status_enum_t oms_getBoolean(const char* cref, bool* value);
```
3.1.28 getBus

Gets the bus object.

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

3.1.29 getComponentType

Gets the type of the given component.

```c
oms_status_enu_t oms_getComponentType(const char* cref, oms_component_enum_t* type);
```

3.1.30 getConnections

Get list of all connections from a given component.

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

3.1.31 getConnector

Gets the connector object of the given connector cref.

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

3.1.32 getElement

Get element information of a given component reference.

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

3.1.33 getElements

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

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

3.1.34 getFMUInfo

Returns FMU specific information.

```c
oms_status_enu_t oms_getFMUInfo(const char* cref, const oms_fmu_info_t** fmuInfo);
```
3.1.35 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.

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

3.1.36 getInteger

Get integer value of given signal.

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

3.1.37 getModelState

Gets the model state of the given model cref.

```c
oms_status_enum_t oms_getModelState(const char* cref, oms_modelState_enum_t* ModelState);
```

3.1.38 getReal

Get real value.

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

3.1.39 getResultFile

Gets the result filename and buffer size of the given model cref.

```c
oms_status_enum_t oms_getResultFile(const char* cref, char** filename, int* bufferSize);
```

3.1.40 getSignalFilter

Gets the signal filter of the given model cref.

```c
oms_status_enum_t oms_getSignalFilter(const char* cref, char** regex);
```

3.1.41 getSolver

Gets the selected solver method of the given system.

```c
oms_status_enum_t oms_getSolver(const char* cref, oms_solver_enum_t* solver);
```
3.1.42 getStartTime

Get the start time from the model.

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

3.1.43 getStopTime

Get the stop time from the model.

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

3.1.44 getSubModelPath

Returns the path of a given component.

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

3.1.45 getSystemType

Gets the type of the given system.

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

3.1.46 getTLMBus

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

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

3.1.47 getTLMVariableTypes

Gets the type of an TLM variable.

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

3.1.48 getTolerance

Gets the tolerance of a given system or component.

```c
oms_status_enu_t oms_getTolerance(const char* cref, double* absoluteTolerance, double* relativeTolerance);
```
3.1.49 getVariableStepSize

Gets the step size parameters.

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

3.1.50 get_version

Returns the library’s version string.

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

3.1.51 importFile

Imports a composite model from a SSP file.

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

3.1.52 importSnapshot

Loads a snapshot to restore a previous model state. The model must be in virgin model state, which means it must not be instantiated.

```c
oms_status_enu_t oms_importSnapshot(const char* cref, const char* snapshot);
```

3.1.53 initialize

Initializes a composite model.

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

3.1.54 instantiate

Instantiates a given composite model.

```c
oms_status_enu_t omsInstantiate(const char* cref);
```

3.1.55 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.

3.1. C-API
3.1.56 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.

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

3.1.57 loadSnapshot

Loads a snapshot to restore a previous model state. The model must be in virgin model state, which means it must not be instantiated.

```c
oms_status_enu_t oms_loadSnapshot(const char* cref, const char* snapshot);
```

3.1.58 newModel

Creates a new and yet empty composite model.

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

3.1.59 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.

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

3.1.60 removeSignalsFromResults

[deprecated: setSignalFilter is the recommended API]

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

```c
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.
3.1.61 rename

Renames a model, system, or component.

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

3.1.62 reset

Reset the composite model after a simulation run.

The FMUs go into the same state as after instantiation.

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

3.1.63 setActivationRatio

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

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

3.1.64 setBoolean

Sets the value of a given boolean signal.

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

3.1.65 setBusGeometry

```c
oms_status_enu_t oms_setBusGeometry(const char* bus, const ssd_connector_˓→geometry_t* geometry);
```

3.1.66 setCommandLineOption

Sets special flags.

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

Available flags:

<table>
<thead>
<tr>
<th>info: Usage: OMSimulator [Options] [Lua script] [FMU] [SSP file] Options: --addParametersToCSV=&lt;arg&gt; Export parameters to .csv file (true, [false]) --clearAllOptions Reset all flags to default values --deleteTempFiles=&lt;bool&gt; Deletes temp files as soon as they are no longer needed ([true], false)</th>
<th></th>
</tr>
</thead>
</table>

(continues on next page)
---emitEvents=<bool> Specifies whether events should be emitted or not ([true], false)
--exportParametersInline=<arg> Export ParameterBindings inline with .ssd file,
--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 (true, [false])
--inputExtrapolation=<bool> Enables input extrapolation
--fetchAllVars=<arg> Workaround for certain FMUs that do not update all internal dependencies automatically
--ignoreInitialUnknowns=<bool> Ignore the initial unknowns from the modelDescription.xml (true, [false])
--inputExtrapolation=<bool> Enables input extrapolation
--communication points ([arg > 1])
--logFile=<arg> [less than 1] Specifies the logfile (stdout is used if no log file is specified)
--logLevel=<int> 0 default, 1 debug, 2 debug+trace
--maxEventIteration=<int> Specifies the max. number of
--maxLoopIteration=<int> Specifies the max. number of
--mode=<arg> [less than 1] Forces a certain FMI mode iff the FMU provides cs and me (cs, [me])
--numProcs=<int> [less than 1] Specifies the max. number of
--progressBar=<bool> Shows a progress bar for the simulation progress in the terminal (true, [false])
--realTime=<bool> Experimental feature for (soft)
--resultFile=<arg> [less than 1] Specifies the name of the output result file
--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 (true, [false])
--suppressPath=<bool> Supresses path information in info messages; especially useful for testing (true, [false])
--tempDir=<arg> Specifies the temp directory
--timeout=<int> Specifies the maximum allowed time (0 disables)
--time in seconds for running a simulation
--tolerance=<double> Specifies the relative tolerance
--version [-v] Displays version information
--wallTime=<bool> Add wall time information for
--workingDir=<arg> Specifies the working directory

---interval=<int> Specifies the number of communication points (arg > 1)
is used if no log file is specified
--logFile=<arg> [less than 1] Specifies the logfile (stdout is used if no log file is specified)
--logLevel=<int> 0 default, 1 debug, 2 debug+trace
--maxEventIteration=<int> Specifies the max. number of
--maxLoopIteration=<int> Specifies the max. number of
--mode=<arg> [less than 1] Forces a certain FMI mode iff the FMU provides cs and me (cs, [me])
--numProcs=<int> [less than 1] Specifies the max. number of
--progressBar=<bool> Shows a progress bar for the simulation progress in the terminal (true, [false])
--realTime=<bool> Experimental feature for (soft)
--resultFile=<arg> [less than 1] Specifies the name of the output result file
--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 (true, [false])
--suppressPath=<bool> Supresses path information in info messages; especially useful for testing (true, [false])
--tempDir=<arg> Specifies the temp directory
--timeout=<int> Specifies the maximum allowed time (0 disables)
--time in seconds for running a simulation
--tolerance=<double> Specifies the relative tolerance
--version [-v] Displays version information
--wallTime=<bool> Add wall time information for
--workingDir=<arg> Specifies the working directory
3.1.67 setConnectionGeometry

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

3.1.68 setConnectorGeometry

Set geometry information to a given connector.

```c
oms_status_enu_t oms_setConnectorGeometry(const char* cref, const ssd_connector_geometry_t* geometry);
```

3.1.69 setElementGeometry

Set geometry information to a given component.

```c
oms_status_enu_t oms_setElementGeometry(const char* cref, const ssd_element_geometry_t* geometry);
```

3.1.70 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.

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

3.1.71 setInteger

Sets the value of a given integer signal.

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

3.1.72 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.

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

3.1.73 setLoggingCallback

Sets a callback function for the logging system.

```c
void oms_setLoggingCallback(void (*cb)(oms_message_type_enu_t type, const char* message));
```
3.1.74 setLoggingInterval

Set the logging interval of the simulation.

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

3.1.75 setLoggingLevel

Enables/Disables debug logging (logDebug and logTrace).

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

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

3.1.76 setMaxLogFileSize

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

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

3.1.77 setReal

Sets the value of a given real signal.

```c
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.

3.1.78 setRealInputDerivative

Sets the first order derivative of a real input signal.

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

```c
oms_status_enu_t oms_setRealInputDerivative(const char* cref, double value);
```
3.1.79 **setResultFile**

Set the result file of the simulation.

```c
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.

3.1.80 **setSignalFilter**

This function specifies the signal filter. The signal filter is used to determine which signals will eventually be exported to the result file.

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

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

3.1.81 **setSolver**

Sets the solver method for the given system.

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

3.1.82 **setStartTime**

Set the start time of the simulation.

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

3.1.83 **setStopTime**

Set the stop time of the simulation.

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

3.1.84 **setTLMBusGeometry**

```c
oms_status_enu_t oms_setTLMBusGeometry(const char* bus, const ssd::connector_geometry_t* geometry);
```

3.1.85 **setTLMConnectionParameters**

Simulates a composite model in its own thread.
3.1.86 setTLMPositionAndOrientation

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

```c
oms_status_enu_t oms_setTLMPositionAndOrientation(const char * cref, double x1, double x2, double x3, double A11, double A12, double A13, double A21, double A22, double A23, double A31, double A32, double A33);
```

3.1.87 setTLMSocketData

Sets data for TLM socket communication.

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

3.1.88 setTempDirectory

Set new temp directory.

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

3.1.89 setTolerance

Sets the tolerance for a given model or system.

```c
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:

```c
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.
3.1.90 setVariableStepSize

Sets the step size parameters for methods with stepsize control.

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

3.1.91 setWorkingDirectory

Set a new working directory.

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

3.1.92 simulate

Simulates a composite model.

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

3.1.93 simulate_asynchronous

Simulates a composite model in its own thread.

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

3.1.94 simulate_realtime

Experimental feature for (soft) real-time simulation.

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

3.1.95 stepUntil

Simulates a composite model until a given time value.

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

3.1.96 terminate

Terminates a given composite model.

```c
oms_status_enu_t oms_terminate(const char* cref);
```
4.1 Examples

```lua
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)
omsetStart("model")
omsetReal("model.root.system1.x_start", 2.5)
omsimulate("model")
omterminate("model")
omdelete("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.

\[
\text{status} = \text{oms_addBus}(\text{cref})
\]

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”.

\[
\text{status} = \text{oms_addConnection}(\text{crefA}, \text{crefB})
\]

The two arguments \(crefA\) and \(crefB\) get swapped automatically if necessary.

4.2.4 addConnector

Adds a connector to a given component.

\[
\text{status} = \text{oms_addConnector}(\text{cref}, \text{causality}, \text{type})
\]

The second argument "causality", should be any of the following,

- \text{oms_causality_input}
- \text{oms_causality_output}
- \text{oms_causality_parameter}
- \text{oms_causality_bidir}
- \text{oms_causality_undefined}

The third argument "type", should be any of the following,

- \text{oms_signal_type_real}
- \text{oms_signal_type_integer}
- \text{oms_signal_type_boolean}
- \text{oms_signal_type_string}
- \text{oms_signal_type_enum}
- \text{oms_signal_type_bus}

4.2.5 addConnectorToBus

Adds a connector to a bus.

\[
\text{status} = \text{oms_addConnectorToBus}(\text{busCref}, \text{connectorCref})
\]

4.2.6 addConnectorToTLMBus

Adds a connector to a TLM bus.

\[
\text{status} = \text{oms_addConnectorToTLMBus}(\text{busCref}, \text{connectorCref}, \text{type})
\]
4.2.7 addExternalModel

Adds an external model to a TLM system.

\[
\text{status} = \text{oms\_addExternalModel}(\text{cref}, \text{path}, \text{startscript})
\]

4.2.8 addSignalsToResults

[deprecated: setSignalFilter is the recommended API]

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

\[
\text{status} = \text{oms\_addSignalsToResults}(\text{cref}, \text{regex})
\]

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.

\[
\text{status} = \text{oms\_addSubModel}(\text{cref}, \text{fmuPath})
\]

4.2.10 addSystem

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

\[
\text{status} = \text{oms\_addSystem}(\text{cref}, \text{type})
\]

4.2.11 addTLMBus

Adds a TLM bus.

\[
\text{status} = \text{oms\_addTLMBus}(\text{cref}, \text{domain}, \text{dimensions}, \text{interpolation})
\]

The second argument "domain", should be any of the following,

- oms_tlm_domain_input
- oms_tlm_domain_output
- oms_tlm_domain_mechanical
- oms_tlm_domain_rotation
- oms_tlm_domain_hydraulic
- oms_tlm_domain_electric

The fourth argument "interpolation", should be any of the following,

- oms_tlm_no_interpolation
- oms_tlm_coarse_grained
- oms_tlm_fine_grained
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:

<table>
<thead>
<tr>
<th>Input</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filenameA</td>
<td>String</td>
<td>Name of first result file to compare.</td>
</tr>
<tr>
<td>filenameB</td>
<td>String</td>
<td>Name of second result file to compare.</td>
</tr>
<tr>
<td>var</td>
<td>String</td>
<td>Name of signal to compare.</td>
</tr>
<tr>
<td>relTol</td>
<td>Number</td>
<td>Relative tolerance.</td>
</tr>
<tr>
<td>absTol</td>
<td>Number</td>
<td>Absolute tolerance.</td>
</tr>
</tbody>
</table>

The following table describes the return values:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>1 if the signal is considered as equal, 0 otherwise</td>
</tr>
</tbody>
</table>

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, event, simulation)
```

4.2.22 exportSSVTemplate

Exports all signals that have start values of one or multiple FMUs to a SSV file that are read from modelDescription.xml. The function can be called for a top level model or a certain FMU component. If called for a top level model, start values of all FMUs are exported to the SSV file. If called for a component, start values of just this FMU are exported to the SSV file.

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

4.2.23 exportSnapshot

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.
4.2.24 extractFMIKind

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

# not available

4.2.25 faultInjection

Defines a new fault injection block.

```lua
status = oms_faultInjection(cref, type, value)
```

<table>
<thead>
<tr>
<th>type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oms_fault_type_bias</td>
<td>( y = y.$original + ) faultValue</td>
</tr>
<tr>
<td>oms_fault_type_gain</td>
<td>( y = y.$original \times ) faultValue</td>
</tr>
<tr>
<td>oms_fault_type_const</td>
<td>( y = ) faultValue</td>
</tr>
</tbody>
</table>

4.2.26 freeMemory

Free the memory allocated by some other API. Pass the object for which memory is allocated. This function is neither needed nor available from the Lua interface.

4.2.27 getBoolean

Get boolean value of given signal.

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

4.2.28 getBus

Gets the bus object.

# not available

4.2.29 getComponentType

Gets the type of the given component.

# not available
4.2.30 getConnections

Get list of all connections from a given component.

```lua
# not available
```

4.2.31 getConnector

Gets the connector object of the given connector cref.

```lua
# not available
```

4.2.32 getElement

Get element information of a given component reference.

```lua
# not available
```

4.2.33 getElements

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

```lua
# not available
```

4.2.34 getFMUInfo

Returns FMU specific information.

```lua
# not available
```

4.2.35 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.

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

4.2.36 getInteger

Get integer value of given signal.

```lua
value, status = oms_getInteger(cref)
```
4.2.37 getModelState

Gets the model state of the given model cref.

```lua
modelState, status = oms_getModelState(cref)
```

4.2.38 getReal

Get real value.

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

4.2.39 getResultFile

Gets the result filename and buffer size of the given model cref.

```lua
# not available
```

4.2.40 getSignalFilter

Gets the signal filter of the given model cref.

```lua
# not available
```

4.2.41 getSolver

Gets the selected solver method of the given system.

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

4.2.42 getStartTime

Get the start time from the model.

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

4.2.43 getStopTime

Get the stop time from the model.

```lua
stopTime, status = oms_getStopTime(cref)
```
4.2.44 getSubModelPath

Returns the path of a given component.

```lua
# not available
```

4.2.45 getSystemType

Gets the type of the given system.

```lua
type, status = oms_getSystemType(cref)
```

4.2.46 getTLMBus

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

```lua
# not available
```

4.2.47 getTLMVariableTypes

Gets the type of a TLM variable.

```lua
# not available
```

4.2.48 getTolerance

Gets the tolerance of a given system or component.

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

4.2.49 getVariableStepSize

Gets the step size parameters.

```lua
initialStepSize, minimumStepSize, maximumStepSize, status = oms_getVariableStepSize(cref)
```

4.2.50 getVersion

Returns the library’s version string.

```lua
version = oms_getVersion()
```
4.2.51 importFile

Imports a composite model from a SSP file.

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

4.2.52 importSnapshot

Loads a snapshot to restore a previous model state. The model must be in virgin model state, which means it must not be instantiated.

```lua
status = oms_importSnapshot(cref, snapshot)
```

4.2.53 initialize

Initializes a composite model.

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

4.2.54 instantiate

Instantiates a given composite model.

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

4.2.55 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.

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

4.2.56 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.

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

4.2.57 loadSnapshot

Loads a snapshot to restore a previous model state. The model must be in virgin model state, which means it must not be instantiated.
4.2.58 newModel

Creates a new and yet empty composite model.

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

4.2.59 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.

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

4.2.60 removeSignalsFromResults

[deprecated: `setSignalFilter` is the recommended API]

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

```lua
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.

4.2.61 rename

Renames a model, system, or component.

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

4.2.62 reset

Reset the composite model after a simulation run.

The FMUs go into the same state as after instantiation.

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

4.2.63 setActivationRatio

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

Sets the value of a given boolean signal.

\[
\text{status} = \text{oms\_setBoolean}(\text{cref}, \text{value})
\]

4.2.65 setBusGeometry

# not available

4.2.66 setCommandLineOption

Sets special flags.

\[
\text{status} = \text{oms\_setCommandLineOption}(\text{cmd})
\]

Available flags:

```
info: Usage: OMSimulator [Options] [Lua script] [FMU] [SSP file]
Options:
  --addParametersToCSV=<arg>        Export parameters to .csv file
  (true, [false])
  --clearAllOptions                Reset all flags to default
  --values                        
  --deleteTempFiles=<bool>         Deletes temp files as soon as
  (true, [false])
  --they are no longer needed      
  --emitEvents=<bool>              Specifies whether events should
  (true, [false])
  --be emitted or not             
  --exportParametersInline=<arg>   Export ParameterBindings inline
  with .ssd file,
  --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
  (true, [false])
  --from the modelDescription.xml
  --inputExtrapolation=<bool>      Enables input extrapolation
  (true, [false])
  --using derivative information
  --intervals=<int> [-i]           Specifies the number of
  (arg > 1)
  --logFile=<arg> [-l]            Specifies the logfile (stdout
  (arg > 1)
  --is used if no log file is specified
  --logLevel=<int>                0 default, 1 debug, 2
  --communication points
  --maxEventIteration=<int>       Specifies the max. number of
  --maxLoopIteration=<int>        Specifies the max. number of
  --iterations for handling a single event
  --iterations for solving algebraic loops between system-level components
  --Internal algebraic loops of components are not affected.
  --mode=<arg> [-m]               Forces a certain FMI mode iff
  (continues on next page)
```
(continued from previous page)

---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 (true, [false])
--realTime=<bool> Experimental feature for real-time co-simulation (true, [false])
--resultFile=<arg> [-r] Specifies the name of the output result file
--setInputDerivatives=<bool> Deprecated; see '--inputExtrapolation'
--skipCSVHeader=<arg> Skip exporting the scv delimiter in the header (true, [false]),
--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 (true, [false])
--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 (true, [false])
--suppressPath=<bool> Supresses path information in info messages; especially useful for testing (true, [false])
--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 (true, [false])
--workingDir=<arg> Specifies the working directory

4.2.67 setConnectionGeometry

# not available

4.2.68 setConnectorGeometry

Set geometry information to a given connector.

# not available

4.2.69 setElementGeometry

Set geometry information to a given component.

# not available

4.2. Lua Scripting Commands
4.2.70  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)

4.2.71  setInteger

Sets the value of a given integer signal.

status = oms_setInteger(cref, value)

4.2.72  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)

4.2.73  setLoggingCallback

Sets a callback function for the logging system.

# not available

4.2.74  setLoggingInterval

Set the logging interval of the simulation.

status = oms_setLoggingInterval(cref, loggingInterval)

4.2.75  setLoggingLevel

Enables/Disables debug logging (logDebug and logTrace).
0 default, 1 default+debug, 2 default+debug+trace

oms_setLoggingLevel(logLevel)

4.2.76  setMaxLogFileSize

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

oms_setMaxLogFileSize(size)
4.2.77 setReal

Sets the value of a given real signal.

\[
\text{status} = \text{oms_setReal}(\text{cref}, \text{value})
\]

This function can be called in different model states:

- Before instantiation: \text{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.

4.2.78 setRealInputDerivative

Sets the first order derivative of a real input signal.

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

\[
\text{status} = \text{oms_setRealInputDerivative}(\text{cref}, \text{value})
\]

4.2.79 setResultFile

Set the result file of the simulation.

\[
\text{status} = \text{oms_setResultFile}(\text{cref}, \text{filename})
\]

\[
\text{status} = \text{oms_setResultFile}(\text{cref}, \text{filename}, \text{bufferSize})
\]

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

4.2.80 setSignalFilter

This function specifies the signal filter. The signal filter is used to determine which signals will eventually be exported to the result file.

\[
\text{status} = \text{oms_setSignalFilter}(\text{cref}, \text{regex})
\]

The second argument, i.e. \text{regex}, is a regular expression (C++11). “.*” and “(.)*” can be used to hit all variables.

4.2.81 setSolver

Sets the solver method for the given system.

\[
\text{status} = \text{oms_setSolver}(\text{cref}, \text{solver})
\]
<table>
<thead>
<tr>
<th>solver</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oms_solver_sc_explicit_euler</td>
<td>sc-system</td>
<td>Explicit euler with fixed step size</td>
</tr>
<tr>
<td>oms_solver_sc_cvode</td>
<td>sc-system</td>
<td>CVODE with adaptive stepsize</td>
</tr>
<tr>
<td>oms_solver_wc_ma</td>
<td>wc-system</td>
<td>default master algorithm with fixed step size</td>
</tr>
<tr>
<td>oms_solver_wc_mav</td>
<td>wc-system</td>
<td>master algorithm with adaptive stepsize</td>
</tr>
<tr>
<td>oms_solver_wc_mav2</td>
<td>wc-system</td>
<td>master algorithm with adaptive stepsize (double-step)</td>
</tr>
</tbody>
</table>

### 4.2.82 setStartTime

Set the start time of the simulation.

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

### 4.2.83 setStopTime

Set the stop time of the simulation.

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

### 4.2.84 setTLMBusGeometry

# not available

### 4.2.85 setTLMConnectionParameters

Simulates a composite model in its own thread.

# not available

### 4.2.86 setTLMPositionAndOrientation

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

```lua
status = oms_setTLMPositionAndOrientation(cref, x1, x2, x3, A11, A12, A13, ...
A21, A22, A23, A31, A32, A33)
```

### 4.2.87 setTLMSocketData

Sets data for TLM socket communication.

```lua
status = oms_setTLMSocketData(cref, address, managerPort, monitorPort)
```
4.2.88 setTempDirectory

Set new temp directory.

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

4.2.89 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:

```
omsetolerance("model", absoluteTolerance, relativeTolerance);
omsetolerance("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.

4.2.90 setVariableStepSize

Sets the step size parameters for methods with stepsize control.

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

4.2.91 setWorkingDirectory

Set a new working directory.

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

4.2.92 simulate

Simulates a composite model.

```
status = oms_simulate(cref)
```
4.2.93 simulate_asynchronous

Simulates a composite model in its own thread.

```
# not available
```

4.2.94 simulate_realtime

Experimental feature for (soft) real-time simulation.

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

4.2.95 stepUntil

Simulates a composite model until a given time value.

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

4.2.96 terminate

Terminates a given composite model.

```
status = oms_terminate(cref)
```
This is a shared library that provides a Python interface for the OMSimulatorLib library.

### 5.1 Examples

```python
from OMSimulator import OMSimulator

oms = 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.
5.2.2 addBus

Adds a bus to a given component.

```python
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”.

```python
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.

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

The second argument "causality", should be any of the following,

- oms.input
- oms.output
- oms.parameter
- oms.bidir
- oms.undefined

The third argument "type", should be any of the following,

- oms.signal_type_real
- oms.signal_type_integer
- oms.signal_type_boolean
- oms.signal_type_string
- oms.signal_type_enum
- oms.signal_type_bus

5.2.5 addConnectorToBus

Adds a connector to a bus.

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

5.2.6 addConnectorToTLMBus

Adds a connector to a TLM bus.
5.2.7 addExternalModel

Adds an external model to a TLM system.

\[
\text{status} = \text{oms.addExternalModel}(\text{cref}, \text{path}, \text{startscript})
\]

5.2.8 addSignalsToResults

[deprecated: \textit{setSignalFilter} is the recommended API]

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

\[
\text{status} = \text{oms.addSignalsToResults}(\text{cref}, \text{regex})
\]

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

5.2.9 addSubModel

Adds a component to a system.

\[
\text{status} = \text{oms.addSubModel}(\text{cref}, \text{fmuPath})
\]

5.2.10 addSystem

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

\[
\text{status} = \text{oms.addSystem}(\text{cref}, \text{type})
\]

5.2.11 addTLMBus

Adds a TLM bus.

\[
\text{status} = \text{oms.addTLMBus}(\text{cref}, \text{domain}, \text{dimensions}, \text{interpolation})
\]

The second argument \texttt{"domain"}, should be any of the following,

\begin{verbatim}
oms.tlm\_domain\_input
oms.tlm\_domain\_output
oms.tlm\_domain\_mechanical
oms.tlm\_domain\_rotational
oms.tlm\_domain\_hydraulic
oms.tlm\_domain\_electric
\end{verbatim}

The fourth argument \texttt{"interpolation"}, should be any of the following,

(continues on next page)
5.2.12 addTLMConnection

Connects two TLM connectors.

\[
\text{status} = \text{oms.addTLMConnection(crefA, crefB, delay, alpha, linearimpedance, angularimpedance)}
\]

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.

\[
\text{oms.compareSimulationResults(filenameA, filenameB, var, relTol, absTol)}
\]

The following table describes the input values:

<table>
<thead>
<tr>
<th>Input</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filenameA</td>
<td>String</td>
<td>Name of first result file to compare.</td>
</tr>
<tr>
<td>filenameB</td>
<td>String</td>
<td>Name of second result file to compare.</td>
</tr>
<tr>
<td>var</td>
<td>String</td>
<td>Name of signal to compare.</td>
</tr>
<tr>
<td>relTol</td>
<td>Number</td>
<td>Relative tolerance.</td>
</tr>
<tr>
<td>absTol</td>
<td>Number</td>
<td>Absolute tolerance.</td>
</tr>
</tbody>
</table>

The following table describes the return values:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>1 if the signal is considered as equal, 0 otherwise</td>
</tr>
</tbody>
</table>

5.2.15 copySystem

Copies a system.

\[
\text{status} = \text{oms.copySystem(source, target)}
\]
5.2.16 delete

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

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

5.2.17 deleteConnection

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

```python
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.

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

5.2.19 deleteConnectorFromTLMBus

Deletes a connector from a given TLM bus.

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

5.2.20 export

Exports a composite model to a SPP file.

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

5.2.21 exportDependencyGraphs

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

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

5.2.22 exportSSVTemplate

Exports all signals that have start values of one or multiple FMUs to a SSV file that are read from modelDescription.xml. The function can be called for a top level model or a certain FMU component. If called for a top level model, start values of all FMUs are exported to the SSV file. If called for a component, start values of just this FMU are exported to the SSV file.
status = oms.exportSSVTemplate(cref, filename)

### 5.2.23 exportSnapshot

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.

content, status = oms.exportSnapshot(cref)

### 5.2.24 extractFMIKind

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

# not available

### 5.2.25 faultInjection

Defines a new fault injection block.

status = oms.faultInjection(cref, type, value)

<table>
<thead>
<tr>
<th>type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oms_fault_type_bias</td>
<td>( y = y.$original + \text{faultValue} )</td>
</tr>
<tr>
<td>oms_fault_type_gain</td>
<td>( y = y.$original * \text{faultValue} )</td>
</tr>
<tr>
<td>oms_fault_type_const</td>
<td>( y = \text{faultValue} )</td>
</tr>
</tbody>
</table>

### 5.2.26 freeMemory

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

oms.freeMemory(obj)

### 5.2.27 getBoolean

Get boolean value of given signal.

value, status = oms.getBoolean(cref)

### 5.2.28 getBus

Gets the bus object.
5.2.29 getComponentType

Gets the type of the given component.

# not available

5.2.30 getConnections

Get list of all connections from a given component.

# not available

5.2.31 getConnector

Gets the connector object of the given connector cref.

# not available

5.2.32 getElement

Get element information of a given component reference.

# not available

5.2.33 getElements

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

# not available

5.2.34 getFMUInfo

Returns FMU specific information.

# not available

5.2.35 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)
5.2.36 getInteger
Get integer value of given signal.

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

5.2.37 getModelState
Gets the model state of the given model cref.

```python
# not available
```

5.2.38 getReal
Get real value.

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

5.2.39 getResultFile
Gets the result filename and buffer size of the given model cref.

```python
# not available
```

5.2.40 getSignalFilter
Gets the signal filter of the given model cref.

```python
# not available
```

5.2.41 getSolver
Gets the selected solver method of the given system.

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

5.2.42 getStartTime
Get the start time from the model.

```python
startTime, status = oms.getStartTime(cref)
```
5.2.43 getStopTime

Get the stop time from the model.

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

5.2.44 getSubModelPath

Returns the path of a given component.

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

5.2.45 getSystemType

Gets the type of the given system.

```python
type, status = oms.getSystemType(cref)
```

5.2.46 getTLMBus

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

```python
# not available
```

5.2.47 getTLMVariableTypes

Gets the type of an TLM variable.

```python
# not available
```

5.2.48 getTolerance

Gets the tolerance of a given system or component.

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

5.2.49 getVariableStepSize

Gets the step size parameters.

```python
initialStepSize, minimumStepSize, maximumStepSize, status = oms.
getVariableStepSize(cref)
```
5.2.50  getVersion

Returns the library’s version string.

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

5.2.51  importFile

Imports a composite model from a SSP file.

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

5.2.52  importSnapshot

Loads a snapshot to restore a previous model state. The model must be in virgin model state, which means it must not be instantiated.

```python
status = oms.importSnapshot(cref, snapshot)
```

5.2.53  initialize

Initializes a composite model.

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

5.2.54  instantiate

Instantiates a given composite model.

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

5.2.55  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.

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

5.2.56  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.
5.2.57 loadSnapshot

Loads a snapshot to restore a previous model state. The model must be in virgin model state, which means it must not be instantiated.

```python
status = oms.loadSnapshot(cref, snapshot)
```

5.2.58 newModel

Creates a new and yet empty composite model.

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

5.2.59 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.

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

5.2.60 removeSignalsFromResults

[deprecated: `setSignalFilter` is the recommended API]

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

```python
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.

5.2.61 rename

 Renames a model, system, or component.

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

5.2.62 reset

Reset the composite model after a simulation run.

The FMUs go into the same state as after instantiation.
status = oms.reset(cref)

### 5.2.63 setActivationRatio

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

# not yet available

### 5.2.64 setBoolean

Sets the value of a given boolean signal.

status = oms.setBoolean(cref, value)

### 5.2.65 setBusGeometry

# not available

### 5.2.66 setCommandLineOption

Sets special flags.

status = oms.setCommandLineOption(cmd)

Available flags:

<table>
<thead>
<tr>
<th>info: Usage: OMSimulator [Options] [Lua script] [FMU] [SSP file]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Options:</td>
</tr>
<tr>
<td>--addParametersToCSV=&lt;arg&gt; Export parameters to .csv file</td>
</tr>
<tr>
<td>(true, [false])</td>
</tr>
<tr>
<td>--clearAllOptions Reset all flags to default</td>
</tr>
<tr>
<td>--values</td>
</tr>
<tr>
<td>--deleteTempFiles=&lt;bool&gt; Deletes temp files as soon as they are no longer needed ([true], false)</td>
</tr>
<tr>
<td>--emitEvents=&lt;bool&gt; Specifies whether events should be emitted or not ([true], false)</td>
</tr>
<tr>
<td>--exportParametersInline=&lt;arg&gt; Export ParameterBindings inline</td>
</tr>
<tr>
<td>--with .ssd file,</td>
</tr>
<tr>
<td>--fetchAllVars=&lt;arg&gt; Workaround for certain FMUs that do not update all internal dependencies automatically</td>
</tr>
<tr>
<td>--help [-h] Displays the help text</td>
</tr>
<tr>
<td>--ignoreInitialUnknowns=&lt;bool&gt; Ignore the initial unknowns from the modelDescription.xml ([true], [false])</td>
</tr>
<tr>
<td>--inputExtrapolation=&lt;bool&gt; Enables input extrapolation</td>
</tr>
<tr>
<td>--outputEvents=&lt;bool&gt; Specifies the number of using derivative information ([true], [false])</td>
</tr>
<tr>
<td>--intervals=&lt;int&gt; [-i] Specifies the number of communication points (arg &gt; 1)</td>
</tr>
<tr>
<td>--logFile=&lt;arg&gt; [-l] Specifies the logfile (stdout is used if no log file is specified)</td>
</tr>
<tr>
<td>Option</td>
</tr>
<tr>
<td>---------------------------------------------</td>
</tr>
<tr>
<td>--logLevel=&lt;int&gt;</td>
</tr>
<tr>
<td>--maxEventIteration=&lt;int&gt;</td>
</tr>
<tr>
<td>--maxLoopIteration=&lt;int&gt;</td>
</tr>
<tr>
<td>--iterations for handling a single event</td>
</tr>
<tr>
<td>--iterations for solving algebraic loops</td>
</tr>
<tr>
<td>--Internal algebraic loops of components are not affected.</td>
</tr>
<tr>
<td>--mode=&lt;arg&gt; [-m]</td>
</tr>
<tr>
<td>--numProcs=&lt;int&gt; [-n]</td>
</tr>
<tr>
<td>--progressBar=&lt;bool&gt;</td>
</tr>
<tr>
<td>--realTime=&lt;bool&gt;</td>
</tr>
<tr>
<td>--resultFile=&lt;arg&gt; [-r]</td>
</tr>
<tr>
<td>--setInputDerivatives=&lt;bool&gt;</td>
</tr>
<tr>
<td>--skipCSVHeader=&lt;arg&gt;</td>
</tr>
<tr>
<td>--solver=&lt;arg&gt;</td>
</tr>
<tr>
<td>--solverStats=&lt;bool&gt;</td>
</tr>
<tr>
<td>--startTime=&lt;double&gt; [-s]</td>
</tr>
<tr>
<td>--stopTime=&lt;double&gt; [-t]</td>
</tr>
<tr>
<td>--stripRoot=&lt;bool&gt;</td>
</tr>
<tr>
<td>--suppressPath=&lt;bool&gt;</td>
</tr>
<tr>
<td>--tempDir=&lt;arg&gt;</td>
</tr>
<tr>
<td>--timeout=&lt;int&gt;</td>
</tr>
<tr>
<td>--time in seconds for running a simulation</td>
</tr>
<tr>
<td>--version [-v]</td>
</tr>
<tr>
<td>--wallTime=&lt;bool&gt;</td>
</tr>
<tr>
<td>--workingDir=&lt;arg&gt;</td>
</tr>
</tbody>
</table>

---

5.2.67 setConnectionGeometry

# not available

5.2.68 setConnectorGeometry

Set geometry information to a given connector.

# not available

5.2. Python Scripting Commands
5.2.69 setElementGeometry

Set geometry information to a given component.

# not available

5.2.70 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.

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

5.2.71 setInteger

Sets the value of a given integer signal.

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

5.2.72 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.

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

5.2.73 setLoggingCallback

Sets a callback function for the logging system.

# not available

5.2.74 setLoggingInterval

Set the logging interval of the simulation.

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

5.2.75 setLoggingLevel

Enables/Disables debug logging (logDebug and logTrace).
0 default, 1 default+debug, 2 default+debug+trace

```python
oms.setLoggingLevel(logLevel)
```
5.2.76 setMaxLogFileSize

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

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

5.2.77 setReal

Sets the value of a given real signal.

```python
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.

5.2.78 setRealInputDerivative

Sets the first order derivative of a real input signal.

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

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

5.2.79 setResultFile

Set the result file of the simulation.

```python
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.

5.2.80 setSignalFilter

This function specifies the signal filter. The signal filter is used to determine which signals will eventually be exported to the result file.

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

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

5.2. Python Scripting Commands
5.2.81 setSolver

Sets the solver method for the given system.

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

<table>
<thead>
<tr>
<th>solver</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oms.solver_sc_explicit_euler</td>
<td>sc-system</td>
<td>Explicit euler with fixed step size</td>
</tr>
<tr>
<td>oms.solver_sc_cvode</td>
<td>sc-system</td>
<td>CVODE with adaptive stepsize</td>
</tr>
<tr>
<td>oms.solver_wc_ma</td>
<td>wc-system</td>
<td>default master algorithm with fixed step size</td>
</tr>
<tr>
<td>oms.solver_wc_mav</td>
<td>wc-system</td>
<td>master algorithm with adaptive stepsize</td>
</tr>
<tr>
<td>oms.solver_wc_mav2</td>
<td>wc-system</td>
<td>master algorithm with adaptive stepsize (double-step)</td>
</tr>
</tbody>
</table>

5.2.82 setStartTime

Set the start time of the simulation.

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

5.2.83 setStopTime

Set the stop time of the simulation.

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

5.2.84 setTLMBusGeometry

# not available

5.2.85 setTLMConnectionParameters

Simulates a composite model in its own thread.

# not available

5.2.86 setTLMPositionAndOrientation

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

# not yet available
5.2.87 setTLMSocketData

Sets data for TLM socket communication.

```
# not yet available
```

5.2.88 setTempDirectory

Set new temp directory.

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

5.2.89 setTolerance

Sets the tolerance for a given model or system.

```python
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:

```python
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.

5.2.90 setVariableStepSize

Sets the step size parameters for methods with stepsize control.

```python
status = oms.getVariableStepSize(cref, initialStepSize, minimumStepSize,
                     -> maximumStepSize)
```

5.2.91 setWorkingDirectory

Set a new working directory.

```python
status = oms.setWorkingDirectory(newWorkingDir)
```
5.2.92 simulate

Simulates a composite model.

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

5.2.93 simulate_asynchronous

Simulates a composite model in its own thread.

```python
# not available
```

5.2.94 simulate_realtime

Experimental feature for (soft) real-time simulation.

```python
# not yet available
```

5.2.95 stepUntil

Simulates a composite model until a given time value.

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

5.2.96 terminate

Terminates a given composite model.

```python
status = oms.terminate(cref)
```
CHAPTER SIX

OPENMODELICASCRIPTING

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

6.1 Examples

```c
loadOMSimulator();
oms_setTempDirectory("./temp/");
oms_newModel("model");
oms_addSystem("model.root", OpenModelica.Scripting.oms_system.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");
unloadOMSimulator();
```

6.2 OpenModelica Scripting Commands

6.2.1 RunFile

Simulates a single FMU or SSP model.
6.2.2 addBus

Adds a bus to a given component.

```plaintext
status := oms_addBus(cref);
```

6.2.3 addConnection

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

```plaintext
status := oms_addConnection(crefA, crefB);
```

The two arguments \texttt{crefA} and \texttt{crefB} get swapped automatically if necessary.

6.2.4 addConnector

Adds a connector to a given component.

```plaintext
status := oms_addConnector(cref, causality, type);
```

The second argument \texttt{"causality"}, should be any of the following,

"OpenModelica.Scripting.oms_causality.oms_causality_input"
"OpenModelica.Scripting.oms_causality.oms_causality_output"
"OpenModelica.Scripting.oms_causality.oms_causality_parameter"
"OpenModelica.Scripting.oms_causality.oms_causality_bidir"
"OpenModelica.Scripting.oms_causality.oms_causality_undefined"

The third argument \texttt{type}, should be any of the following,

"OpenModelica.Scripting.oms_signal_type.oms_signal_type_real"
"OpenModelica.Scripting.oms_signal_type.oms_signal_type_integer"
"OpenModelica.Scripting.oms_signal_type.oms_signal_type_boolean"
"OpenModelica.Scripting.oms_signal_type.oms_signal_type_string"
"OpenModelica.Scripting.oms_signal_type.oms_signal_type_enum"
"OpenModelica.Scripting.oms_signal_type.oms_signal_type_bus"

6.2.5 addConnectorToBus

Adds a connector to a bus.

```plaintext
status := oms_addConnectorToBus(busCref, connectorCref);
```

6.2.6 addConnectorToTLMBus

Adds a connector to a TLM bus.
6.2.7 addExternalModel

Adds an external model to a TLM system.

```c
status := oms_addExternalModel(cref, path, startscript);
```

6.2.8 addSignalsToResults

[deprecated: setSignalFilter is the recommended API]

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

```c
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.

6.2.9 addSubModel

Adds a component to a system.

```c
status := oms_addSubModel(cref, fmuPath);
```

6.2.10 addSystem

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

```c
status := oms_addSystem(cref, type);
```

The second argument `type`, should be any of the following,

"OpenModelica.Scripting.oms_system.oms_system_none"
"OpenModelica.Scripting.oms_system.oms_system_tlm"
"OpenModelica.Scripting.oms_system.oms_system_sc"
"OpenModelica.Scripting.oms_system.oms_system_wc"

6.2.11 addTLMBus

Adds a TLM bus.

```c
status := oms_addTLMBus(cref, domain, dimensions, interpolation);
```

The second argument "domain", should be any of the following,

"OpenModelica.Scripting.oms_tlm_domain.oms_tlm_domain_input"
"OpenModelica.Scripting.oms_tlm_domain.oms_tlm_domain_output"

(continues on next page)
The fourth argument "interpolation", should be any of the following,

- "OpenModelica.Scripting.oms_tlm_interpolation.oms_tlm_no_interpolation"
- "OpenModelica.Scripting.oms_tlm_interpolation.oms_tlm_coarse_grained"
- "OpenModelica.Scripting.oms_tlm_interpolation.oms_tlm_fine_grained"

### 6.2.12 addTLMConnection

Connects two TLM connectors.

```plaintext
status := oms_addTLMConnection(crefA, crefB, delay, alpha, linearimpedance,
                                 angularimpedance);
```

### 6.2.13 cancelSimulation_asynchronous

Cancels a running asynchronous simulation.

```plaintext
status := oms_cancelSimulation_asynchronous(cref);
```

### 6.2.14 compareSimulationResults

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

```plaintext
status := oms_compareSimulationResults(filenameA, filenameB, var, relTol, absTol);
```

The following table describes the input values:

<table>
<thead>
<tr>
<th>Input</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filenameA</td>
<td>String</td>
<td>Name of first result file to compare.</td>
</tr>
<tr>
<td>filenameB</td>
<td>String</td>
<td>Name of second result file to compare.</td>
</tr>
<tr>
<td>var</td>
<td>String</td>
<td>Name of signal to compare.</td>
</tr>
<tr>
<td>relTol</td>
<td>Number</td>
<td>Relative tolerance.</td>
</tr>
<tr>
<td>absTol</td>
<td>Number</td>
<td>Absolute tolerance.</td>
</tr>
</tbody>
</table>

The following table describes the return values:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>1 if the signal is considered as equal, 0 otherwise</td>
</tr>
</tbody>
</table>
6.2.15 copySystem

Copies a system.

\[
\text{status} := \text{oms\_copySystem(source, target)};
\]

6.2.16 delete

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

\[
\text{status} := \text{oms\_delete(cref)};
\]

6.2.17 deleteConnection

Deletes the connection between connectors \text{crefA} and \text{crefB}.

\[
\text{status} := \text{oms\_deleteConnection(crefA, crefB)};
\]

The two arguments \text{crefA} and \text{crefB} get swapped automatically if necessary.

6.2.18 deleteConnectorFromBus

Deletes a connector from a given bus.

\[
\text{status} := \text{oms\_deleteConnectorFromBus(busCref, connectorCref)};
\]

6.2.19 deleteConnectorFromTLMBus

Deletes a connector from a given TLM bus.

\[
\text{status} := \text{oms\_deleteConnectorFromTLMBus(busCref, connectorCref)};
\]

6.2.20 export

Exports a composite model to a SPP file.

\[
\text{status} := \text{oms\_export(cref, filename)};
\]

6.2.21 exportDependencyGraphs

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

\[
\text{status} := \text{oms\_exportDependencyGraphs(cref, initialization, event, simulation)};
\]
6.2.22 exportSSVTemplate

Exports all signals that have start values of one or multiple FMUs to a SSV file that are read from modelDescription.xml. The function can be called for a top level model or a certain FMU component. If called for a top level model, start values of all FMUs are exported to the SSV file. If called for a component, start values of just this FMU are exported to the SSV file.

```plaintext
# not available
```

6.2.23 exportSnapshot

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.

```plaintext
(contents, status) := oms_exportSnapshot(cref);
```

6.2.24 extractFMIKind

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

```plaintext
(kind, status) := oms_extractFMIKind(filename);
```

6.2.25 faultInjection

Defines a new fault injection block.

```plaintext
status := oms_faultInjection(cref, type, value);
```

The second argument type, can be any of the following described below

- "OpenModelica.Scripting.oms_fault_type.oms_fault_type_bias"
- "OpenModelica.Scripting.oms_fault_type.oms_fault_type_gain"
- "OpenModelica.Scripting.oms_fault_type.oms_fault_type_const"

<table>
<thead>
<tr>
<th>type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oms_fault_type_bias</td>
<td>y = y.$original + faultValue</td>
</tr>
<tr>
<td>oms_fault_type_gain</td>
<td>y = y.$original * faultValue</td>
</tr>
<tr>
<td>oms_fault_type_const</td>
<td>y = faultValue</td>
</tr>
</tbody>
</table>

6.2.26 freeMemory

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

This function is not needed for OpenModelicaScripting Interface
6.2.27 getBoolean

Get boolean value of given signal.

\[(value, status) := \text{oms\_getBoolean}(\text{cref});\]

6.2.28 getBus

Gets the bus object.

# not available

6.2.29 getComponentType

Gets the type of the given component.

# not available

6.2.30 getConnections

Get list of all connections from a given component.

# not available

6.2.31 getConnector

Gets the connector object of the given connector cref.

# not available

6.2.32 getElement

Get element information of a given component reference.

# not available

6.2.33 getElements

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

# not available
6.2.34 getFMUInfo

Returns FMU specific information.

# not available

6.2.35 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);

6.2.36 getInteger

Get integer value of given signal.

(value, status) := oms_getInteger(cref);

6.2.37 getModelState

Gets the model state of the given model cref.

(modelState, status) := oms_getModelState(cref);

6.2.38 getReal

Get real value.

(value, status) := oms_getReal(cref);

6.2.39 getResultFile

Gets the result filename and buffer size of the given model cref.

# not available

6.2.40 getSignalFilter

Gets the signal filter of the given model cref.

# not available
6.2.41 getSolver

Gets the selected solver method of the given system.

\[(\text{solver}, \text{status}) := \text{oms\_getSolver(cref);}\]

6.2.42 getStartTime

Get the start time from the model.

\[(\text{startTime}, \text{status}) := \text{oms\_getStartTime(cref);}\]

6.2.43 getStopTime

Get the stop time from the model.

\[(\text{stopTime}, \text{status}) := \text{oms\_getStopTime(cref);}\]

6.2.44 getSubModelPath

Returns the path of a given component.

\[(\text{path}, \text{status}) := \text{oms\_getSubModelPath(cref);}\]

6.2.45 getSystemType

Gets the type of the given system.

\[(\text{type}, \text{status}) := \text{oms\_getSystemType(cref);}\]

6.2.46 getTLMBus

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

# not available

6.2.47 getTLMVariableTypes

Gets the type of an TLM variable.

# not available
6.2.48 getTolerance

Gets the tolerance of a given system or component.

\[(\text{absoluteTolerance}, \text{relativeTolerance}, \text{status}) := \text{oms}_\text{getTolerance}(\text{cref});\]

6.2.49 getVariableStepSize

Gets the step size parameters.

\[(\text{initialStepSize}, \text{minimumStepSize}, \text{maximumStepSize}, \text{status}) := \text{oms}_\text{getVariableStepSize}(\text{cref});\]

6.2.50 getVersion

Returns the library’s version string.

\[\text{version} := \text{oms}_\text{getVersion}();\]

6.2.51 importFile

Imports a composite model from a SSP file.

\[(\text{cref}, \text{status}) := \text{oms}_\text{importFile}(\text{filename});\]

6.2.52 importSnapshot

Loads a snapshot to restore a previous model state. The model must be in virgin model state, which means it must not be instantiated.

\[\text{status} := \text{oms}_\text{importSnapshot}(\text{cref}, \text{snapshot});\]

6.2.53 initialize

Initializes a composite model.

\[\text{status} := \text{oms}_\text{initialize}(\text{cref});\]

6.2.54 instantiate

Instantiates a given composite model.

\[\text{status} := \text{oms}_\text{instantiate}(\text{cref});\]
6.2.55 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.

```c
(contents, status) := oms_list(cref);
```

6.2.56 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.

```c
(contents, status) := oms_listUnconnectedConnectors(cref);
```

6.2.57 loadSnapshot

Loads a snapshot to restore a previous model state. The model must be in virgin model state, which means it must not be instantiated.

```c
status := oms_loadSnapshot(cref, snapshot);
```

6.2.58 newModel

Creates a new and yet empty composite model.

```c
status := oms_newModel(cref);
```

6.2.59 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.

```c
(ident, status) := oms_parseModelName(contents);
```

6.2.60 removeSignalsFromResults

[deprecated: setSignalFilter is the recommended API]
Removes all variables that match the given regex to the result file.

```c
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.
6.2.61 rename

Renames a model, system, or component.

\[
\text{status} := \text{oms\_rename}(\text{cref}, \text{newCref});
\]

6.2.62 reset

Reset the composite model after a simulation run.

The FMUs go into the same state as after instantiation.

\[
\text{status} := \text{oms\_reset}(\text{cref});
\]

6.2.63 setActivationRatio

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

# not yet available

6.2.64 setBoolean

Sets the value of a given boolean signal.

\[
\text{status} := \text{oms\_setBoolean}(\text{cref}, \text{value});
\]

6.2.65 setBusGeometry

# not available

6.2.66 setCommandLineOption

Sets special flags.

\[
\text{status} := \text{oms\_setCommandLineOption}(\text{cmd});
\]

Available flags:

<table>
<thead>
<tr>
<th>info:</th>
<th>Usage: OMSimulator [Options] [Lua script] [FMU] [SSP file]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Options:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>--addParametersToCSV=&lt;arg&gt; Export parameters to .csv file</td>
</tr>
<tr>
<td></td>
<td>(true, [false])</td>
</tr>
<tr>
<td></td>
<td>--clearAllOptions Reset all flags to default</td>
</tr>
<tr>
<td></td>
<td>--values</td>
</tr>
<tr>
<td></td>
<td>--deleteTempFiles=bool Deletes temp files as soon as</td>
</tr>
<tr>
<td></td>
<td>(true), false</td>
</tr>
<tr>
<td></td>
<td>--emitEvents=bool Specifies whether events should</td>
</tr>
<tr>
<td></td>
<td>(true), false</td>
</tr>
<tr>
<td></td>
<td>(true), false</td>
</tr>
<tr>
<td>Argument</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>--exportParametersInline=&lt;arg&gt;</td>
<td>Export ParameterBindings inline</td>
</tr>
<tr>
<td>with .ssd file,</td>
<td></td>
</tr>
<tr>
<td>--fetchAllVars=&lt;arg&gt;</td>
<td>Workaround for certain FMUs</td>
</tr>
<tr>
<td>that do not update all internal</td>
<td></td>
</tr>
<tr>
<td>dependencies automatically</td>
<td></td>
</tr>
<tr>
<td>--help [-h]</td>
<td>Displays the help text</td>
</tr>
<tr>
<td>--ignoreInitialUnknowns=&lt;bool&gt;</td>
<td>Ignore the initial unknowns</td>
</tr>
<tr>
<td>from the modelDescription.xml (true,</td>
<td></td>
</tr>
<tr>
<td>[false])</td>
<td></td>
</tr>
<tr>
<td>--inputExtrapolation=&lt;bool&gt;</td>
<td>Enables input extrapolation</td>
</tr>
<tr>
<td>using derivative information (true,</td>
<td></td>
</tr>
<tr>
<td>[false])</td>
<td></td>
</tr>
<tr>
<td>--intervals=&lt;int&gt; [-i]</td>
<td>Specifies the number of communication points (arg &gt; 1)</td>
</tr>
<tr>
<td>--logFile=&lt;arg&gt; [-l]</td>
<td>Specifies the logfile (stdout is used if no log file is specified)</td>
</tr>
<tr>
<td>--is used if no log file is specified</td>
<td></td>
</tr>
<tr>
<td>--logLevel=&lt;int&gt; 0 default, 1 debug,</td>
<td>Specifies the loglevel (debug+trace</td>
</tr>
<tr>
<td>--maxEventIteration=&lt;int&gt;</td>
<td>Specifies the max. number of maxEventIteration=&lt;int&gt;</td>
</tr>
<tr>
<td>--maxLoopIteration=&lt;int&gt;</td>
<td>Specifies the max. number of maxLoopIteration=&lt;int&gt;</td>
</tr>
<tr>
<td>--mode=&lt;arg&gt; [-m]</td>
<td>Forces a certain FMI mode if mode=&lt;arg&gt;</td>
</tr>
<tr>
<td>--numProcs=&lt;int&gt; 0=default, 1=default</td>
<td>Specifies the max. number of numProcs=&lt;int&gt;</td>
</tr>
<tr>
<td>--processors to use (0=auto, 1=default)</td>
<td>Shows a progress bar for the processors to use (0=auto, 1=default)</td>
</tr>
<tr>
<td>--progressBar=&lt;bool&gt;</td>
<td></td>
</tr>
<tr>
<td>--realTime=&lt;bool&gt;</td>
<td>Experimental feature for real-time co-simulation (true, [false])</td>
</tr>
<tr>
<td>--realTime co-simulation (true, [false])</td>
<td></td>
</tr>
<tr>
<td>--resultFile=&lt;arg&gt; [-r]</td>
<td>Specifies the name of the resultFile=&lt;arg&gt;</td>
</tr>
<tr>
<td>--setInputDerivatives=&lt;bool&gt;</td>
<td>Deprecated; see --setInputDerivatives=&lt;bool&gt;</td>
</tr>
<tr>
<td>--solver=&lt;arg&gt; euler, [cvode]</td>
<td>Specifies the integration method</td>
</tr>
<tr>
<td>--solverStats=&lt;bool&gt;</td>
<td>Adds solver stats to the result file, e.g. step size; not supported for all solvers (true, [false])</td>
</tr>
<tr>
<td>--skipCSVHeader=&lt;arg&gt;</td>
<td>Skip exporting the scv</td>
</tr>
<tr>
<td>--skipCSVHeader=&lt;arg&gt;</td>
<td></td>
</tr>
<tr>
<td>--delimeter in the header (true, [false]),</td>
<td></td>
</tr>
<tr>
<td>--solver=&lt;arg&gt; euler, [cvode]</td>
<td>Specifies the integration method</td>
</tr>
<tr>
<td>--method (euler, [cvode])</td>
<td></td>
</tr>
<tr>
<td>--solverStats=&lt;bool&gt;</td>
<td></td>
</tr>
<tr>
<td>--file, e.g. step size; not supported for all solvers (true, [false])</td>
<td>Specifies the start time</td>
</tr>
<tr>
<td>--startTime=&lt;double&gt; [-s]</td>
<td>Specifies the start time</td>
</tr>
<tr>
<td>--stopTime=&lt;double&gt; [-t]</td>
<td>Specifies the stop time</td>
</tr>
<tr>
<td>--stripRoot=&lt;bool&gt;</td>
<td>Removes the root system prefix</td>
</tr>
<tr>
<td>--stripRoot=&lt;bool&gt;</td>
<td></td>
</tr>
<tr>
<td>--from all exported signals (true, [false])</td>
<td>Supresses path information in --from all exported signals (true, [false])</td>
</tr>
<tr>
<td>--suppressPath=&lt;bool&gt;</td>
<td></td>
</tr>
<tr>
<td>--info messages; especially useful for testing (true, [false])</td>
<td>Specifies the temp directory</td>
</tr>
<tr>
<td>--tempDir=&lt;arg&gt;</td>
<td>Specifies the maximum allowed time in seconds for running a simulation (0 disables)</td>
</tr>
<tr>
<td>--tempDir=&lt;arg&gt;</td>
<td>Specifies the maximum allowed time in seconds for running a simulation (0 disables)</td>
</tr>
<tr>
<td>--timeout=&lt;int&gt;</td>
<td>Displays version information</td>
</tr>
<tr>
<td>--time in seconds for running a</td>
<td>Add wall time information for running a simulation (0 disables)</td>
</tr>
<tr>
<td>simulation (0 disables)</td>
<td>Specifies the relative tolerance</td>
</tr>
<tr>
<td>--tolerance=&lt;double&gt;</td>
<td></td>
</tr>
<tr>
<td>--version [-v]</td>
<td></td>
</tr>
<tr>
<td>--wallTime=&lt;bool&gt;</td>
<td></td>
</tr>
<tr>
<td>--to the result file (true, [false])</td>
<td>Specifies the working directory</td>
</tr>
<tr>
<td>--workingDir=&lt;arg&gt;</td>
<td></td>
</tr>
</tbody>
</table>
6.2.67 setConnectionGeometry

# not available

6.2.68 setConnectorGeometry

Set geometry information to a given connector.

# not available

6.2.69 setElementGeometry

Set geometry information to a given component.

# not available

6.2.70 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);
```

6.2.71 setInteger

Sets the value of a given integer signal.

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

6.2.72 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);
```

6.2.73 setLoggingCallback

Sets a callback function for the logging system.

# not available
6.2.74 setLoggingInterval

Set the logging interval of the simulation.

\[
\text{status} := \text{oms_setLoggingInterval}(\text{cref}, \text{loggingInterval});
\]

6.2.75 setLoggingLevel

Enables/Disables debug logging (logDebug and logTrace).

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

\[
\text{oms_setLoggingLevel} \left( \text{logLevel} \right);
\]

6.2.76 setMaxLogFileSize

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

# not available

6.2.77 setReal

Sets the value of a given real signal.

\[
\text{status} := \text{oms_setReal}(\text{cref}, \text{value});
\]

This function can be called in different model states:

- Before instantiation: \text{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.

6.2.78 setRealInputDerivative

Sets the first order derivative of a real input signal.

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

\[
\text{status} := \text{oms_setRealInputDerivative}(\text{cref}, \text{value});
\]

6.2.79 setResultFile

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

### 6.2.80 setSignalFilter

This function specifies the signal filter. The signal filter is used to determine which signals will eventually be exported to the result file.

```c
status := oms_setSignalFilter(cref, regex);
```

The second argument, i.e. regex, is a regular expression (C++11). 

".*" and "(.)*" can be used to hit all variables.

### 6.2.81 setSolver

Sets the solver method for the given system.

```c
status := oms_setSolver(cref, solver);
```

The second argument "solver" should be any of the following,

"OpenModelica.Scripting.oms_solver.oms_solver_none"
"OpenModelica.Scripting.oms_solver.oms_solver_sc_min"
"OpenModelica.Scripting.oms_solver.oms_solver_sc_explicit_euler"
"OpenModelica.Scripting.oms_solver.oms_solver_sc_cvode"
"OpenModelica.Scripting.oms_solver.oms_solver_sc_max"
"OpenModelica.Scripting.oms_solver.oms_solver_wc_min"
"OpenModelica.Scripting.oms_solver.oms_solver_wc_max"
"OpenModelica.Scripting.oms_solver.oms_solver_wc_assc"
"OpenModelica.Scripting.oms_solver.oms_solver_wc_mav2"
"OpenModelica.Scripting.oms_solver.oms_solver_wc_max"

### 6.2.82 setStartTime

Set the start time of the simulation.

```c
status := oms_setStartTime(cref, startTime);
```

### 6.2.83 setStopTime

Set the stop time of the simulation.

```c
status := oms_setStopTime(cref, stopTime);
```
6.2.84 setTLMBusGeometry

# not available

6.2.85 setTLMConnectionParameters

Simulates a composite model in its own thread.

# not available

6.2.86 setTLMPositionAndOrientation

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

\[
\text{status} := \text{oms_setTLMPositionAndOrientation}(\text{cref}, x_1, x_2, x_3, A_{11}, A_{12}, A_{13}, \rightarrow A_{21}, A_{22}, A_{23}, A_{31}, A_{32}, A_{33});
\]

6.2.87 setTLMSocketData

Sets data for TLM socket communication.

\[
\text{status} := \text{oms_setTLMSocketData}(\text{cref}, \text{address}, \text{managerPort}, \text{monitorPort});
\]

6.2.88 setTempDirectory

Set new temp directory.

\[
\text{status} := \text{oms_setTempDirectory}(\text{newTempDir});
\]

6.2.89 setTolerance

Sets the tolerance for a given model or system.

\[
\text{status} := \text{oms_setTolerance}(\text{const char* cref}, \text{double tolerance});
\text{status} := \text{oms_setTolerance}(\text{const char* cref}, \text{double absoluteTolerance,.double relativeTolerance});
\]

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:

\[
\text{oms_setTolerance("model", absoluteTolerance, relativeTolerance);}
\text{oms_setTolerance("model.root", absoluteTolerance, relativeTolerance);}
\]

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.

### 6.2.90 setVariableStepSize

Sets the step size parameters for methods with stepsize control.

```c
status := oms_getVariableStepSize(cref, initialStepSize, minimumStepSize, maximumStepSize);
```

### 6.2.91 setWorkingDirectory

Set a new working directory.

```c
status := oms_setWorkingDirectory(newWorkingDir);
```

### 6.2.92 simulate

Simulates a composite model.

```c
status := oms_simulate(cref);
```

### 6.2.93 simulate_asynchronous

Simulates a composite model in its own thread.

```
# not available
```

### 6.2.94 simulate_realtime

Experimental feature for (soft) real-time simulation.

```
# not yet available
```

### 6.2.95 stepUntil

Simulates a composite model until a given time value.

```c
status := oms_stepUntil(cref, stopTime);
```
6.2.96 terminate

Terminates a given composite model.

\[
\text{status := oms_terminate(cref);}
\]
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.

See also FMI documentation.

7.1 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
7.2 Add System

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

![Fig. 4: OMEdit: Add System](image)

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

![Fig. 5: OMEdit: Add System Dialog](image)

7.3 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](image)

**7.4 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](image)

That will pop-up a dialog to enter various simulation settings.
Fig. 9: OMEdit: Instantiate Model Dialog

7.5 Simulate

Select the simulate item from the OMSimulator menu.
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
<?xml version="1.0" encoding="UTF-8"?>
<ssd:SystemStructureDescription name="Test" version="Draft20180219">
  <ssd:Elements>
    <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>
        </ssd:System>
      </ssd:Elements>
    </ssd:System>
  </ssd:Elements>
</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.
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, \texttt{wc1} and \texttt{wc2}. System \texttt{wc1} contains two TLM connectors, one of type 1D signal and one of type 1D mechanical. System \texttt{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
<?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 name="bus2" type="tlm" domain="mechanical" dimension="1" interpolation="none"/>
        </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>
        </ssd:Annotation>
      </ssd:Annotations>
    </ssd:System>
  </ssd:Elements>
</ssd:System>
```

(continues on next page)
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

<table>
<thead>
<tr>
<th>tlmType</th>
<th>causality</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;value&quot;</td>
<td>input/output</td>
</tr>
</tbody>
</table>

1D physical (no interpolation)

<table>
<thead>
<tr>
<th>tlmType</th>
<th>causality</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;state&quot;</td>
<td>output</td>
</tr>
<tr>
<td>&quot;flow&quot;</td>
<td>output</td>
</tr>
<tr>
<td>&quot;effort&quot;</td>
<td>input</td>
</tr>
</tbody>
</table>

1D physical (coarse-grained interpolation)

<table>
<thead>
<tr>
<th>tlmType</th>
<th>causality</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;state&quot;</td>
<td>output</td>
</tr>
<tr>
<td>&quot;flow&quot;</td>
<td>output</td>
</tr>
<tr>
<td>&quot;wave&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;impedance&quot;</td>
<td>input</td>
</tr>
</tbody>
</table>

1D physical (fine-grained interpolation)
### TLM Connections

<table>
<thead>
<tr>
<th>tlmType</th>
<th>causality</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;state&quot;</td>
<td>output</td>
</tr>
<tr>
<td>&quot;flow&quot;</td>
<td>output</td>
</tr>
<tr>
<td>&quot;wave1&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave2&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave3&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave4&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave5&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave6&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave7&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave8&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave9&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave10&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;time1&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;time2&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;time3&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;time4&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;time5&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;time6&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;time7&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;time8&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;time9&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;time10&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;impedance&quot;</td>
<td>input</td>
</tr>
</tbody>
</table>

**3D physical (no interpolation)**
### 3D physical (coarse-grained interpolation)

<table>
<thead>
<tr>
<th>tlmType</th>
<th>causality</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;state1&quot;</td>
<td>output</td>
</tr>
<tr>
<td>&quot;state2&quot;</td>
<td>output</td>
</tr>
<tr>
<td>&quot;state3&quot;</td>
<td>output</td>
</tr>
<tr>
<td>&quot;A11&quot;</td>
<td>output</td>
</tr>
<tr>
<td>&quot;A12&quot;</td>
<td>output</td>
</tr>
<tr>
<td>&quot;A13&quot;</td>
<td>output</td>
</tr>
<tr>
<td>&quot;A21&quot;</td>
<td>output</td>
</tr>
<tr>
<td>&quot;A22&quot;</td>
<td>output</td>
</tr>
<tr>
<td>&quot;A23&quot;</td>
<td>output</td>
</tr>
<tr>
<td>&quot;A31&quot;</td>
<td>output</td>
</tr>
<tr>
<td>&quot;A32&quot;</td>
<td>output</td>
</tr>
<tr>
<td>&quot;A33&quot;</td>
<td>output</td>
</tr>
<tr>
<td>&quot;flow1&quot;</td>
<td>output</td>
</tr>
<tr>
<td>&quot;flow2&quot;</td>
<td>output</td>
</tr>
<tr>
<td>&quot;flow3&quot;</td>
<td>output</td>
</tr>
<tr>
<td>&quot;flow4&quot;</td>
<td>output</td>
</tr>
<tr>
<td>&quot;flow5&quot;</td>
<td>output</td>
</tr>
<tr>
<td>&quot;flow6&quot;</td>
<td>output</td>
</tr>
<tr>
<td>&quot;effort1&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;effort2&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;effort3&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;effort4&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;effort5&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;effort6&quot;</td>
<td>input</td>
</tr>
</tbody>
</table>
### TLM Connections

**3D physical (fine-grained interpolation)**

<table>
<thead>
<tr>
<th>tlmType</th>
<th>causality</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;state1&quot;</td>
<td>output</td>
</tr>
<tr>
<td>&quot;state2&quot;</td>
<td>output</td>
</tr>
<tr>
<td>&quot;state3&quot;</td>
<td>output</td>
</tr>
<tr>
<td>&quot;A11&quot;</td>
<td>output</td>
</tr>
<tr>
<td>&quot;A12&quot;</td>
<td>output</td>
</tr>
<tr>
<td>&quot;A13&quot;</td>
<td>output</td>
</tr>
<tr>
<td>&quot;A21&quot;</td>
<td>output</td>
</tr>
<tr>
<td>&quot;A22&quot;</td>
<td>output</td>
</tr>
<tr>
<td>&quot;A23&quot;</td>
<td>output</td>
</tr>
<tr>
<td>&quot;A31&quot;</td>
<td>output</td>
</tr>
<tr>
<td>&quot;A32&quot;</td>
<td>output</td>
</tr>
<tr>
<td>&quot;A33&quot;</td>
<td>output</td>
</tr>
<tr>
<td>&quot;flow1&quot;</td>
<td>output</td>
</tr>
<tr>
<td>&quot;flow2&quot;</td>
<td>output</td>
</tr>
<tr>
<td>&quot;flow3&quot;</td>
<td>output</td>
</tr>
<tr>
<td>&quot;flow4&quot;</td>
<td>output</td>
</tr>
<tr>
<td>&quot;flow5&quot;</td>
<td>output</td>
</tr>
<tr>
<td>&quot;flow6&quot;</td>
<td>output</td>
</tr>
<tr>
<td>&quot;wave1&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave2&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave3&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave4&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave5&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave6&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;linearimpedance&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;angularimpedance&quot;</td>
<td>input</td>
</tr>
</tbody>
</table>
Table 1 – continued from previous page

<table>
<thead>
<tr>
<th>tlmType</th>
<th>causality</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;flow6&quot;</td>
<td>output</td>
</tr>
<tr>
<td>&quot;wave1_1&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave1_2&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave1_3&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave1_4&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave1_5&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave1_6&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave2_1&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave2_2&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave2_3&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave2_4&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave2_5&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave2_6&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave3_1&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave3_2&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave3_3&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave3_4&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave3_5&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave3_6&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave4_1&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave4_2&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave4_3&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave4_4&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave4_5&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave4_6&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave5_1&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave5_2&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave5_3&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave5_4&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave5_5&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave5_6&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave6_1&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave6_2&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave6_3&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave6_4&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave6_5&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave6_6&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave7_1&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave7_2&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave7_3&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave7_4&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave7_5&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave7_6&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave8_1&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave8_2&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave8_3&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave8_4&quot;</td>
<td>input</td>
</tr>
</tbody>
</table>

Continued on next page
Table 1 – continued from previous page

<table>
<thead>
<tr>
<th>tlmType</th>
<th>causality</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;wave8_5&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave8_6&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave9_1&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave9_2&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave9_3&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave9_4&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave9_5&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave9_6&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave10_1&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave10_2&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave10_3&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave10_4&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave10_5&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;wave10_6&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;time1&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;time2&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;time3&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;time4&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;time5&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;time6&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;time7&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;time8&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;time9&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;time10&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;linearimpedance&quot;</td>
<td>input</td>
</tr>
<tr>
<td>&quot;angularimpedance&quot;</td>
<td>input</td>
</tr>
</tbody>
</table>
INDEX

O
OMEdit, 77
OMSimulator, 1
  Examples, 4
  Flags, 3
OMSimulatorLib, 4
  C-API, 5
OMSimulatorLua, 21
  Examples, 23
  Scripting Commands, 23
OMSimulatorPython, 40
  Examples, 41
  Scripting Commands, 41
OpenModelicaScripting, 58
  Examples, 59
  Scripting Commands, 59

S
SSP, 84
  Bus connections, 85
  TLM connections, 87
  TLM Systems, 86