OMSimulator Documentation

Release v2.1.0-dev-162-g63d0b36

Lennart Ochel

Apr 29, 2020
CONTENTS

1 Introduction 1

2 OMSimulator 3
   2.1 OMSimulator Flags ............................................. 3
   2.2 Examples .......................................................... 4

3 OMSimulatorLib 5
   3.1 C-API ............................................................. 5

4 OMSimulatorLua 21
   4.1 Examples .......................................................... 21
   4.2 Lua Scripting Commands .......................................... 21

5 OMSimulatorPython 39
   5.1 Examples .......................................................... 39
   5.2 Python Scripting Commands ...................................... 39

6 OpenModelicaScripting 57
   6.1 Examples .......................................................... 57
   6.2 OpenModelica Scripting Commands .............................. 57

7 Graphical Modelling 75
   7.1 New OMSimulator Model ......................................... 75
   7.2 Add System ....................................................... 77
   7.3 Add SubModel .................................................... 77
   7.4 Instantiate Model ............................................... 79
   7.5 Simulate ........................................................... 80

8 OMSysIdent 81
   8.1 Examples .......................................................... 81
   8.2 Lua Scripting Commands .......................................... 83

9 SSP Support 87
   9.1 Bus Connections .................................................. 87
   9.2 TLM Systems ...................................................... 88
   9.3 TLM Connections ................................................ 89

Index 97
CHAPTER ONE

INTRODUCTION

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`:

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--clearAllOptions</code></td>
<td>Reset all flags to default.</td>
</tr>
<tr>
<td><code>--deleteTempFiles=&lt;bool&gt;</code></td>
<td>Deletes temp files as soon as they are no longer needed.</td>
</tr>
<tr>
<td><code>--emitEvents=&lt;bool&gt;</code></td>
<td>Specifies whether events should be emitted or not.</td>
</tr>
<tr>
<td><code>--fetchAllVars=&lt;arg&gt;</code></td>
<td>Workaround for certain FMUs that do not update all internal dependencies automatically</td>
</tr>
<tr>
<td><code>--help [-h]</code></td>
<td>Displays the help text.</td>
</tr>
<tr>
<td><code>--ignoreInitialUnknowns=&lt;bool&gt;</code></td>
<td>Ignore the initial unknowns from the modelDescription.xml (true, [false])</td>
</tr>
<tr>
<td><code>--inputExtrapolation=&lt;bool&gt;</code></td>
<td>Enables input extrapolation using derivative information (true, [false])</td>
</tr>
<tr>
<td><code>--intervals=&lt;int&gt;</code></td>
<td>Specifies the number of communication points (arg &gt; 1)</td>
</tr>
<tr>
<td><code>--logFile=&lt;arg&gt;</code></td>
<td>Specifies the logfile (stdout is used if no log file is specified)</td>
</tr>
<tr>
<td><code>--logLevel=&lt;int&gt;</code></td>
<td>0 default, 1 debug, 2 debug+trace</td>
</tr>
<tr>
<td><code>--maxEventIteration=&lt;int&gt;</code></td>
<td>Specifies the max. number of iterations for handling a single event</td>
</tr>
<tr>
<td><code>--mode=&lt;arg&gt;</code></td>
<td>Forces a certain FMI mode iff the FMU provides cs and me ([cs], me)</td>
</tr>
<tr>
<td><code>--numProcs=&lt;int&gt;</code></td>
<td>Specifies the max. number of processors to use (0=auto, 1=default)</td>
</tr>
<tr>
<td><code>--progressBar=&lt;bool&gt;</code></td>
<td>Shows a progress bar for the simulation progress in the terminal (true, [false])</td>
</tr>
<tr>
<td><code>--realTime=&lt;bool&gt;</code></td>
<td>Experimental feature for (soft) real-time co-simulation (true, [false])</td>
</tr>
<tr>
<td><code>--resultFile=&lt;arg&gt;</code></td>
<td>Specifies the name of the output result file</td>
</tr>
<tr>
<td><code>--solver=&lt;arg&gt;</code></td>
<td>Specifies the integration method (euler, [cvode])</td>
</tr>
</tbody>
</table>

(continues on next page)
### 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

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.
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* simulation);
```

3.1.22 extractFMIKind

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

Defines a new fault injection block.

```c
oms_status_enu_t oms_faultInjection(const char* signal, oms_fault_type_enu_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>

3.1.24 freeMemory

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

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

3.1.25 getBoolean

Get boolean value of given signal.

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

3.1.26 getBus

Gets the bus object.

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

3.1.27 getComponentType

Gets the type of the given component.

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

3.1.28 getConnections

Get list of all connections from a given component.

```c
```
3.1.29 getConnector

Gets the connector object of the given connector cref.

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

3.1.30 getElement

Get element information of a given component reference.

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

3.1.31 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);
```

3.1.32 getFMUInfo

Returns FMU specific information.

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

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

3.1.34 getInteger

Get integer value of given signal.

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

Gets the model state of the given model cref.

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

3.1.36 getReal

Get real value.

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

3.1.37 getSolver

Gets the selected solver method of the given system.

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

3.1.38 getStartTime

Get the start time from the model.

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

3.1.39 getStopTime

Get the stop time from the model.

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

3.1.40 getSubModelPath

Returns the path of a given component.

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

3.1.41 getSystemType

Gets the type of the given system.

```c
oms_status_enu_t oms_getSystemType(const char* cref, oms_system_enu_t* type);
```
3.1.42 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);
```

3.1.43 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);
```

3.1.44 getTolerance

Gets the tolerance of a given system or component.

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

3.1.45 getVariableStepSize

Gets the step size parameters.

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

3.1.46 getVersion

Returns the library’s version string.

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

3.1.47 importFile

Imports a composite model from a SSP file.

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

3.1.48 initialize

Initializes a composite model.

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

Instantiates a given composite model.

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

3.1.50 list

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

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

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

3.1.51 listUnconnectedConnectors

Lists all unconnected connectors of a given system.

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

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

3.1.52 loadSnapshot

Loads a snapshot to restore a previous model state.

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

3.1.53 newModel

Creates a new and yet empty composite model.

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

3.1.54 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 omsParseModelName(const char* contents, char** cref);
```
3.1.55 removeSignalsFromResults

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.

3.1.56 rename

Renames a model, system, or component.

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

3.1.57 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);
```

3.1.58 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);
```

3.1.59 setBoolean

Sets the value of a given boolean signal.

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

3.1.60 setBusGeometry

```c
oms_status_enu_t oms_setBusGeometry(const char* bus, const ssd_connector_geometry_t* geometry);
```

3.1.61 setCommandLineOption

Sets special flags.
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:</th>
</tr>
</thead>
<tbody>
<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>--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 using derivative information ([true], [false])</td>
</tr>
<tr>
<td>--fetchAllVars=&lt;arg&gt; Workaround for certain FMUs that do not update all internal dependencies automatically</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>--logLevel=&lt;int&gt; 0 default, 1 debug, 2 debug+trace</td>
</tr>
<tr>
<td>--maxEventIteration=&lt;int&gt; Specifies the max. number of iterations for handling a single event</td>
</tr>
<tr>
<td>--mode=&lt;arg&gt; [-m] Forces a certain FMI mode iff the FMU provides cs and me ([cs], me)</td>
</tr>
<tr>
<td>--numProcs=&lt;int&gt; [-n] Specifies the max. number of processors to use (0=auto, 1=default)</td>
</tr>
<tr>
<td>--progressBar=&lt;bool&gt; Shows a progress bar for the simulation progress in the terminal (true, [false])</td>
</tr>
<tr>
<td>--realTime=&lt;bool&gt; Experimental feature for (soft) real-time co-simulation (true, [false])</td>
</tr>
<tr>
<td>--resultFile=&lt;arg&gt; [-r] Specifies the name of the result file</td>
</tr>
<tr>
<td>--setInputDerivatives=&lt;bool&gt; Deprecated; see '--inputExtrapolation'</td>
</tr>
<tr>
<td>--solver=&lt;arg&gt; Specifies the integration method (euler, [cvode])</td>
</tr>
<tr>
<td>--solverStats=&lt;bool&gt; Adds solver stats to the result file, e.g. step size; not supported for all solvers ([true], [false])</td>
</tr>
<tr>
<td>--startTime=&lt;double&gt; [-s] Specifies the start time</td>
</tr>
<tr>
<td>--stopTime=&lt;double&gt; [-t] Specifies the stop time</td>
</tr>
<tr>
<td>--stripRoot=&lt;bool&gt; Removes the root system prefix from all exported signals ([true], [false])</td>
</tr>
<tr>
<td>--suppressPath=&lt;bool&gt; Supresses path information in info messages; especially useful for testing ([true], [false])</td>
</tr>
<tr>
<td>--tempDir=&lt;arg&gt; Specifies the temp directory</td>
</tr>
<tr>
<td>--timeout=&lt;int&gt; Specifies the maximum allowed time in seconds for running a simulation (0 disables)</td>
</tr>
<tr>
<td>--version [-v] Displays version information</td>
</tr>
<tr>
<td>--wallTime=&lt;bool&gt; Add wall time information for the result file ([true], [false])</td>
</tr>
<tr>
<td>--workingDir=&lt;arg&gt; Specifies the working directory</td>
</tr>
</tbody>
</table>
3.1.62 setConnectionGeometry

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

3.1.63 setConnectorGeometry

Set geometry information to a given connector.

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

3.1.64 setElementGeometry

Set geometry information to a given component.

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

3.1.65 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);
```

3.1.66 setInteger

Sets the value of a given integer signal.

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

3.1.67 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);
```

3.1.68 setLoggingCallback

Sets a callback function for the logging system.

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

Set the logging interval of the simulation.

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

3.1.70 setLoggingLevel

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

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

3.1.71 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);
```

3.1.72 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.

3.1.73 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);
```
3.1.74 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.

3.1.75 setSignalFilter

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

3.1.76 setSolver

Sets the solver method for the given system.

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

3.1.77 setStartTime

Set the start time of the simulation.

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

3.1.78 setStopTime

Set the stop time of the simulation.

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

3.1.79 setTLMBusGeometry

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

3.1.80 setTLMConnectionParameters

Simulates a composite model in its own thread.

```c
oms_status_enu_t oms_setTLMConnectionParameters(const char* crefA, const_
→char* crefB, const oms_tlm_connection_parameters_t* parameters);
```
### 3.1.81 setTLMPositionAndOrientation

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

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

### 3.1.82 setTLMSocketData

Sets data for TLM socket communication.

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

### 3.1.83 setTempDirectory

Set new temp directory.

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

### 3.1.84 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.

### 3.1.85 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. C-API
3.1.86 setWorkingDirectory

Set a new working directory.

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

3.1.87 simulate

Simulates a composite model.

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

3.1.88 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));
```

3.1.89 simulate_realtime

Experimental feature for (soft) real-time simulation.

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

3.1.90 stepUntil

Simulates a composite model until a given time value.

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

3.1.91 terminate

Terminates a given composite model.

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

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

### 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)

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

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

### 4.2 Lua Scripting Commands

#### 4.2.1 RunFile

Simulates a single FMU or SSP model.

```python
# not available
```
4.2.2 addBus

Adds a bus to a given component.

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

4.2.3 addConnection

Adds a new connection between connectors \( A \) and \( B \). The connectors need to be specified as fully qualified component references, e.g., “model.system.component.signal”.

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

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

4.2.4 addConnector

Adds a connector to a given component.

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

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

- oms_causality_input
- oms_causality_output
- oms_causality_parameter
- oms_causality_bidir
- oms_causality_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

4.2.5 addConnectorToBus

Adds a connector to a bus.

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

4.2.6 addConnectorToTLMBus

Adds a connector to a TLM bus.

```
status = oms_addConnectorToTLMBus(busCref, connectorCref, type)
```
### 4.2.7 addExternalModel

Adds an external model to a TLM system.

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

### 4.2.8 addSignalsToResults

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

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

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

### 4.2.9 addSubModel

Adds a component to a system.

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

### 4.2.10 addSystem

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

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

### 4.2.11 addTLMBus

Adds a TLM bus.

```lua
status = oms_addTLMBus(cref, domain, dimensions, 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_rotational
- 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.

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

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.

\[
\text{oms_compareSimulationResults}(\text{filenameA}, \text{filenameB}, \text{var}, \text{relTol}, \text{absTol})
\]

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.

\[
\text{status} = \text{oms_copySystem}(\text{source}, \text{target})
\]

4.2.16 delete

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

\[
\text{status} = \text{oms_delete}(\text{cref})
\]
4.2.17 deleteConnection

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

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

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

4.2.18 deleteConnectorFromBus

Deletes a connector from a given bus.

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

4.2.19 deleteConnectorFromTLMBus

Deletes a connector from a given TLM bus.

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

4.2.20 export

Exports a composite model to a SPP file.

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

4.2.21 exportDependencyGraphs

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

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

4.2.22 extractFMIKind

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

```
# not available
```

4.2.23 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 + 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>

### 4.2.24 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.

### 4.2.25 getBoolean

Get boolean value of given signal.

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

### 4.2.26 getBus

Gets the bus object.

```
# not available
```

### 4.2.27 getComponentType

Gets the type of the given component.

```
# not available
```

### 4.2.28 getConnections

Get list of all connections from a given component.

```
# not available
```

### 4.2.29 getConnector

Gets the connector object of the given connector cref.

```
# not available
```
4.2.30 getElement

Get element information of a given component reference.

```lua
# not available
```

4.2.31 getElements

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

```lua
# not available
```

4.2.32 getFMUInfo

Returns FMU specific information.

```lua
# not available
```

4.2.33 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)
```

4.2.34 getInteger

Get integer value of given signal.

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

4.2.35 getModelState

Gets the model state of the given model cref.

```lua
# not available
```

4.2.36 getReal

Get real value.

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

Gets the selected solver method of the given system.

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

4.2.38 getStartTime

Get the start time from the model.

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

4.2.39 getStopTime

Get the stop time from the model.

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

4.2.40 getSubModelPath

Returns the path of a given component.

```lua
# not available
```

4.2.41 getSystemType

Gets the type of the given system.

```lua
# not available
```

4.2.42 getTLMBus

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

```lua
# not available
```

4.2.43 getTLMVariableTypes

Gets the type of a TLM variable.

```lua
# not available
```
4.2.44 `getTolerance`

Gets the tolerance of a given system or component.

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

4.2.45 `getVariableStepSize`

Gets the step size parameters.

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

4.2.46 `getVersion`

Returns the library’s version string.

```plaintext
version = oms_getVersion()
```

4.2.47 `importFile`

Imports a composite model from a SSP file.

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

4.2.48 `initialize`

Initializes a composite model.

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

4.2.49 `instantiate`

Instantiates a given composite model.

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

4.2.50 `list`

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

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

```plaintext
contents, status = oms_list(cref)
```
4.2.51 listUnconnectedConnectors

Lists all unconnected connectors of a given system.
Memory is allocated for contents. The caller is responsible to free it using the C-API. The Lua and Python bindings take care of the memory and the caller doesn’t need to call free.

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

4.2.52 loadSnapshot

Loads a snapshot to restore a previous model state.

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

4.2.53 newModel

Creates a new and yet empty composite model.

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

4.2.54 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)
```

4.2.55 removeSignalsFromResults

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.

4.2.56 rename

 Renames a model, system, or component.

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

Reset the composite model after a simulation run.
The FMUs go into the same state as after instantiation.

```c
status = oms_Reset(cref)
```

4.2.58 setActivationRatio

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

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

4.2.59 setBoolean

Sets the value of a given boolean signal.

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

4.2.60 setBusGeometry

### not available

4.2.61 setCommandLineOption

Sets special flags.

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

Available flags:

```
info: Usage: OMSimulator [Options] [Lua script] [FMU] [SSP file]
Options:
  --clearAllOptions Reset all flags to default
  --deleteTempFiles=bool Deletes temp files as soon as they are no longer needed ([true], false)
  --emitEvents=bool Specifies whether events should be emitted or not ([true], false)
  --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
  --intervals=int [-i] Specifies the number of communication points (arg > 1)
```

(continues on next page)
OMSimulator Documentation, Release v2.1.0-dev-162-g63d0b36

(continued from previous page)

```
--logFile=<arg> [-l] 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 iterations for handling a single event
--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])
--realTime=<bool> Experimental feature for (soft) real-time co-simulation (true, [false])
--resultFile=<arg> [-r] 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])
--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
```

4.2.62 setConnectionGeometry

```
# not available
```

4.2.63 setConnectorGeometry

Set geometry information to a given connector.

```
# not available
```

4.2.64 setElementGeometry

Set geometry information to a given component.
### 4.2.65 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.

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

### 4.2.66 setInteger

Sets the value of a given integer signal.

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

### 4.2.67 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.

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

### 4.2.68 setLoggingCallback

Sets a callback function for the logging system.

```lua
# not available
```

### 4.2.69 setLoggingInterval

Set the logging interval of the simulation.

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

### 4.2.70 setLoggingLevel

Enables/Disables debug logging (logDebug and logTrace).

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

```lua
oms_setLoggingLevel(logLevel)
```
4.2.71 setMaxLogFileSize

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

```lua
oms_setMaxLogFileSize(size)
```

4.2.72 setReal

Sets the value of a given real signal.

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

This function can be called in different model states:

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

4.2.73 setRealInputDerivative

Sets the first order derivative of a real input signal.

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

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

4.2.74 setResultFile

Set the result file of the simulation.

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

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

4.2.75 setSignalFilter

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

4.2.76 setSolver

Sets the solver method for the given system.
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>

4.2.77 setStartTime

Set the start time of the simulation.

status = oms_setStartTime(cref, startTime)

4.2.78 setStopTime

Set the stop time of the simulation.

status = oms_setStopTime(cref, stopTime)

4.2.79 setTLMBusGeometry

# not available

4.2.80 setTLMConnectionParameters

Simulates a composite model in its own thread.

# not available

4.2.81 setTLMPositionAndOrientation

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

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

4.2.82 setTLMSocketData

Sets data for TLM socket communication.

status = oms_setTLMSocketData(cref, address, managerPort, monitorPort)
4.2.83 setTempDirectory

Set new temp directory.

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

4.2.84 setTolerance

Sets the tolerance for a given model or system.

```c
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:

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

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

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

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

4.2.85 setVariableStepSize

Sets the step size parameters for methods with stepsize control.

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

4.2.86 setWorkingDirectory

Set a new working directory.

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

4.2.87 simulate

Simulates a composite model.

```c
status = oms_simulate(cref)
```
4.2.88 simulate_asynchronous

Simulates a composite model in its own thread.

```lua
# not available
```

4.2.89 simulate_realtime

Experimental feature for (soft) real-time simulation.

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

4.2.90 stepUntil

Simulates a composite model until a given time value.

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

4.2.91 terminate

Terminates a given composite model.

```lua
status = oms_terminate(cref)
```
5.1 Examples

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

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

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

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

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

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

5.2 Python Scripting Commands

5.2.1 RunFile

Simulates a single FMU or SSP model.

# not available
5.2.2 addBus

Adds a bus to a given component.

```plaintext
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”.

```plaintext
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.

```plaintext
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.

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

5.2.6 addConnectorToTLMBus

Adds a connector to a TLM bus.

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

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.

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 "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.default
- oms.coarsegrained
- oms.finegrained
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.

\[
\text{status} = \text{oms.delete(cref)}
\]
5.2.17 deleteConnection

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

\begin{verbatim}
status = oms.deleteConnection(crefA, crefB)
\end{verbatim}

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

5.2.18 deleteConnectorFromBus

Deletes a connector from a given bus.

\begin{verbatim}
status = oms.deleteConnectorFromBus(busCref, connectorCref)
\end{verbatim}

5.2.19 deleteConnectorFromTLMBus

Deletes a connector from a given TLM bus.

\begin{verbatim}
status = oms.deleteConnectorFromTLMBus(busCref, connectorCref)
\end{verbatim}

5.2.20 export

Exports a composite model to a SPP file.

\begin{verbatim}
status = oms.export(cref, filename)
\end{verbatim}

5.2.21 exportDependencyGraphs

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

\begin{verbatim}
status = oms.exportDependencyGraphs(cref, initialization, simulation)
\end{verbatim}

5.2.22 extractFMIKind

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

\begin{verbatim}
# not available
\end{verbatim}

5.2.23 faultInjection

Defines a new fault injection block.

\begin{verbatim}
status = oms.faultInjection(cref, type, value)
\end{verbatim}
### Table of Fault Types

<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>

#### 5.2.24 freeMemory

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

```python
om.s.freeMemory(obj)
```

#### 5.2.25 getBoolean

Get boolean value of given signal.

```python
c_value, status = oms.getBoolean(cref)
```

#### 5.2.26 getBus

Gets the bus object.

```python
# not available
```

#### 5.2.27 GetComponentType

Gets the type of the given component.

```python
# not available
```

#### 5.2.28 getConnections

Get list of all connections from a given component.

```python
# not available
```

#### 5.2.29 getConnector

Gets the connector object of the given connector cref.

```python
# not available
```
5.2.30 `getElement`

Get element information of a given component reference.

```
# not available
```

5.2.31 `getElements`

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

```
# not available
```

5.2.32 `getFMUInfo`

Returns FMU specific information.

```
# not available
```

5.2.33 `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.

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

5.2.34 `getInteger`

Get integer value of given signal.

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

5.2.35 `getModelState`

Gets the model state of the given model cref.

```
# not available
```

5.2.36 `getReal`

Get real value.

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

Gets the selected solver method of the given system.

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

5.2.38 getStartTime

Get the start time from the model.

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

5.2.39 getStopTime

Get the stop time from the model.

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

5.2.40 getSubModelPath

Returns the path of a given component.

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

5.2.41 getSystemType

Gets the type of the given system.

```
# not available
```

5.2.42 getTLMBus

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

```
# not available
```

5.2.43 getTLMVariableTypes

Gets the type of an TLM variable.

```
# not available
```
5.2.44 getTolerance

Gets the tolerance of a given system or component.

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

5.2.45 getVariableStepSize

Gets the step size parameters.

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

5.2.46 getVersion

Returns the library’s version string.

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

5.2.47 importFile

Imports a composite model from a SSP file.

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

5.2.48 initialize

Initializes a composite model.

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

5.2.49 instantiate

Instantiates a given composite model.

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

5.2.50 list

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

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

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

5.2. Python Scripting Commands
5.2.51 listUnconnectedConnectors

Lists all unconnected connectors of a given system.

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

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

5.2.52 loadSnapshot

Loads a snapshot to restore a previous model state.

```python
# not available
```

5.2.53 newModel

Creates a new and yet empty composite model.

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

5.2.54 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)
```

5.2.55 removeSignalsFromResults

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.

5.2.56 rename

 Renames a model, system, or component.

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

Reset the composite model after a simulation run.

The FMUs go into the same state as after instantiation.

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

**5.2.58 setActivationRatio**

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

```python
# not yet available
```

**5.2.59 setBoolean**

Sets the value of a given boolean signal.

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

**5.2.60 setBusGeometry**

```python
# not available
```

**5.2.61 setCommandLineOption**

Sets special flags.

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

Available flags:

```
info: Usage: OMSimulator [Options] [Lua script] [FMU] [SSP file]
Options:
  --clearAllOptions Reset all flags to default
  --values
    --deleteTempFiles=<bool> Deletes temp files as soon as
      they are no longer needed ([true], false)
    --emitEvents=<bool> Specifies whether events should
      be emitted or not ([true], false)
    --fetchAllVars=<arg> Workaround for certain FMUs
    --help [-h] Displays the help text
    --ignoreInitialUnknowns=<bool> Ignore the initial unknowns
      from the modelDescription.xml (true, [false])
    --inputExtrapolation=<bool> Enables input extrapolation
    --intervals=<int> [-i] Specifies the number of
      communication points (arg > 1)
(continues on next page)
```
--logFile=<arg>  [-l] 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 iterations for handling a single event
--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])
--realTime=<bool> Experimental feature for (soft) real-time co-simulation (true, [false])
--resultFile=<arg>  [-r] Specifies the name of the output result file
--setInputDerivatives=<bool> Deprecated; see '--inputExtrapolation'
--solver=<arg> Specifies the integration method (euler, [cvode])
--solverStats=<bool> Adds solver stats to the result file, e.g. step size; not supported for all solvers (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

5.2.62 setConnectionGeometry

# not available

5.2.63 setConnectorGeometry

Set geometry information to a given connector.

# not available

5.2.64 setElementGeometry

Set geometry information to a given component.
5.2.65 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.

\[
\text{status} = \text{oms.setFixedStepSize}(\text{cref}, \text{stepSize})
\]

5.2.66 setInteger

Sets the value of a given integer signal.

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

5.2.67 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.

\[
\text{status} = \text{oms.setLogFile}(\text{filename})
\]

5.2.68 setLoggingCallback

Sets a callback function for the logging system.

\# not available

5.2.69 setLoggingInterval

Set the logging interval of the simulation.

\[
\text{status} = \text{oms.setLoggingInterval}(\text{cref}, \text{loggingInterval})
\]

5.2.70 setLoggingLevel

Enables/Disables debug logging (logDebug and logTrace).

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

\[
\text{oms.setLoggingLevel}(\text{logLevel})
\]

5.2. Python Scripting Commands
5.2.71 setMaxLogFileSize

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

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

5.2.72 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.

5.2.73 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)
```

5.2.74 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.

5.2.75 setSignalFilter

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

5.2.76 setSolver

Sets the solver method for the given system.
status = oms.setSolver(cref, solver)

#### 5.2.77 setStartTime

Set the start time of the simulation.

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

#### 5.2.78 setStopTime

Set the stop time of the simulation.

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

#### 5.2.79 setTLMBusGeometry

# not available

#### 5.2.80 setTLMConnectionParameters

Simulates a composite model in its own thread.

# not available

#### 5.2.81 setTLMPositionAndOrientation

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

# not yet available

#### 5.2.82 setTLMSocketData

Sets data for TLM socket communication.

# not yet available

#### 5.2.83 setTempDirectory

Set new temp directory.

```python
cstatus = oms.setTempDirectory(newTempDir)
```
5.2.84 setTolerance

Sets the tolerance for a given model or system.

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

Default values are \(10^{-4}\) 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.

5.2.85 setVariableStepSize

Sets the step size parameters for methods with stepsize control.

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

5.2.86 setWorkingDirectory

Set a new working directory.

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

5.2.87 simulate

Simulates a composite model.

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

5.2.88 simulate_asynchronous

Simulates a composite model in its own thread.

```c
# not available
```
5.2.89 simulate_realtime

Experimental feature for (soft) real-time simulation.

```python
# not yet available
```

5.2.90 stepUntil

Simulates a composite model until a given time value.

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

5.2.91 terminate

Terminates a given composite model.

```python
status = oms.terminate(cref)
```
This is a shared library that provides a OpenModelica Scripting interface for the OMSimulatorLib library.

6.1 Examples

```plaintext
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 A and B. The connectors need to be specified as fully qualified component references, e.g., “model.system.component.signal”.

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

The two arguments `crefA` and `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 "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 `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.
status := oms_addConnectorToTLMBus(busCref, connectorCref, type);

6.2.7 addExternalModel

Adds an external model to a TLM system.

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

6.2.8 addSignalsToResults

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

status := oms_addSignalsToResults(cref, regex);

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

6.2.9 addSubModel

Adds a component to a system.

status := oms_addSubModel(cref, fmuPath);

6.2.10 addSystem

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

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.

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"
"OpenModelica.Scripting.oms_tlm_domain.oms_tlm_domain_mechanical"
"OpenModelica.Scripting.oms_tlm_domain.oms_tlm_domain_rotational"

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

"OpenModelica.Scripting.oms_tlm_domain.oms_tlm_domain_hydraulic"
"OpenModelica.Scripting.oms_tlm_domain.oms_tlm_domain_electric"

6.2.12 addTLMConnection

Connects two TLM connectors.

\[
\text{status} := \text{oms_addTLMConnection(crefA, crefB, delay, alpha, linearimpedance, angularimpedance)};
\]

6.2.13 cancelSimulation_asynchronous

Cancels a running asynchronous simulation.

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

6.2.14 compareSimulationResults

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

\[
\text{status} := \text{oms_compareSimulationResults(filenameA, filenameB, var, relTol, absTol)};
\]

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>

6.2.15 copySystem

Copies a system.
6.2.16 delete

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

```
status := oms_delete(cref);
```

6.2.17 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.

6.2.18 deleteConnectorFromBus

Deletes a connector from a given bus.

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

6.2.19 deleteConnectorFromTLMBus

Deletes a connector from a given TLM bus.

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

6.2.20 export

Exports a composite model to a SPP file.

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

6.2.21 exportDependencyGraphs

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

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

6.2.22 extractFMIKind

Extracts the FMI kind of a given FMU from the file system.
(kind, status) := oms_extractFM IKind(filename);

### 6.2.23 faultInjection

Defines a new fault injection block.

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

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.\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>

### 6.2.24 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.

### 6.2.25 getBoolean

Get boolean value of given signal.

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

### 6.2.26 getBus

Gets the bus object.

# not available

### 6.2.27 getComponentType

Gets the type of the given component.

# not available

### 6.2.28 getConnections

Get list of all connections from a given component.
6.2.29 getConnector

Gets the connector object of the given connector cref.

# not available

6.2.30 getElement

Get element information of a given component reference.

# not available

6.2.31 getElements

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

# not available

6.2.32 getFMUInfo

Returns FMU specific information.

# not available

6.2.33 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);

6.2.34 getInteger

Get integer value of given signal.

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

6.2.35 getModelState

Gets the model state of the given model cref.

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

Get real value.

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

6.2.37 getSolver

Gets the selected solver method of the given system.

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

6.2.38 getStartTime

Get the start time from the model.

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

6.2.39 getStopTime

Get the stop time from the model.

```c
(stopTime, status) := oms_getStopTime(cref);
```

6.2.40 getSubModelPath

Returns the path of a given component.

```c
(path, status) := oms_getSubModelPath(cref);
```

6.2.41 getSystemType

Gets the type of the given system.

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

6.2.42 getTLMBus

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

```c
# not available
```
6.2.43 getTLMVariableTypes

Gets the type of an TLM variable.

# not available

6.2.44 getTolerance

Gets the tolerance of a given system or component.

(absoluteTolerance, relativeTolerance, status) := oms_getTolerance(cref);

6.2.45 getVariableStepSize

Gets the step size parameters.

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

6.2.46 getVersion

Returns the library’s version string.

version := oms_getVersion();

6.2.47 importFile

Imports a composite model from a SSP file.

(cref, status) := oms_importFile(filename);

6.2.48 initialize

Initializes a composite model.

status := oms_initialize(cref);

6.2.49 instantiate

Instantiates a given composite model.

status := oms_instantiate(cref);
6.2.50 list

Lists the SSD representation of a given model, system, or component.
Memory is allocated for contents. The caller is responsible to free it using the C-API. The Lua and Python bindings take care of the memory and the caller doesn’t need to call free.

\[(\text{contents, status}) := \text{oms\_list}(\text{cref});\]

6.2.51 listUnconnectedConnectors

Lists all unconnected connectors of a given system.
Memory is allocated for contents. The caller is responsible to free it using the C-API. The Lua and Python bindings take care of the memory and the caller doesn’t need to call free.

\[(\text{contents, status}) := \text{oms\_listUnconnectedConnectors}(\text{cref});\]

6.2.52 loadSnapshot

 Loads a snapshot to restore a previous model state.

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

6.2.53 newModel

 Creates a new and yet empty composite model.

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

6.2.54 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.

\[(\text{ident, status}) := \text{oms\_parseModelName}(\text{contents});\]

6.2.55 removeSignalsFromResults

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

\[\text{status} := \text{oms\_removeSignalsFromResults}(\text{cref}, \text{regex});\]

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

Renames a model, system, or component.

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

6.2.57 reset

Reset the composite model after a simulation run.
The FMUs go into the same state as after instantiation.

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

6.2.58 setActivationRatio

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

# not yet available

6.2.59 setBoolean

Sets the value of a given boolean signal.

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

6.2.60 setBusGeometry

# not available

6.2.61 setCommandLineOption

Sets special flags.

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

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>Reset all flags to default</td>
</tr>
<tr>
<td>~values</td>
<td>Deletes temp files as soon as they are no longer needed ([true], false)</td>
</tr>
<tr>
<td>~be emitted or not ([true], false)</td>
<td>Specifies whether events should be emitted or not</td>
</tr>
<tr>
<td>~that do not update all internal dependencies automatically</td>
<td>Workaround for certain FMUs</td>
</tr>
</tbody>
</table>

(continues on next page)
6.2.62 setConnectionGeometry

# not available

6.2.63 setConnectorGeometry

Set geometry information to a given connector.
6.2.64 setElementGeometry

Set geometry information to a given component.

# not available

6.2.65 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);
```

6.2.66 setInteger

Sets the value of a given integer signal.

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

6.2.67 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);
```

6.2.68 setLoggingCallback

Sets a callback function for the logging system.

# not available

6.2.69 setLoggingInterval

Set the logging interval of the simulation.

```
status := oms_setLoggingInterval(cref, loggingInterval);
```
6.2.70 setLoggingLevel

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

```c
oms_setLoggingLevel(logLevel);
```

6.2.71 setMaxLogFileSize

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

```c
# not available
```

6.2.72 setReal

Sets the value of a given real signal.

```c
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.

6.2.73 setRealInputDerivative

Sets the first order derivative of a real input signal.

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

```c
status := oms_setRealInputDerivative(cref, value);
```

6.2.74 setResultFile

Set the result file of the simulation.

```c
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.
6.2.75 setSignalFilter

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

6.2.76 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_mav"
- "OpenModelica.Scripting.oms_solver.oms_solver_wc_assc"
- "OpenModelica.Scripting.oms_solver.oms_solver_wc_mav2"
- "OpenModelica.Scripting.oms_solver.oms_solver_wc_max"

6.2.77 setStartTime

Set the start time of the simulation.

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

6.2.78 setStopTime

Set the stop time of the simulation.

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

6.2.79 setTLMBusGeometry

# not available

6.2.80 setTLMConnectionParameters

Simulates a composite model in its own thread.

# not available
6.2.81 setTLMPositionAndOrientation

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

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

6.2.82 setTLMSocketData

Sets data for TLM socket communication.

```c
status := oms_setTLMSocketData(cref, address, managerPort, monitorPort);
```

6.2.83 setTempDirectory

Set new temp directory.

```c
status := oms_setTempDirectory(newTempDir);
```

6.2.84 setTolerance

Sets the tolerance for a given model or system.

```c
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:

```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.

6.2.85 setVariableStepSize

Sets the step size parameters for methods with stepsize control.

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

Set a new working directory.

\[
\text{status} := \text{oms\_setWorkingDirectory}(\text{newWorkingDir});
\]

6.2.87 simulate

Simulates a composite model.

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

6.2.88 simulate_asynchronous

Simulates a composite model in its own thread.

# not available

6.2.89 simulate_realtime

Experimental feature for (soft) real-time simulation.

# not yet available

6.2.90 stepUntil

Simulates a composite model until a given time value.

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

6.2.91 terminate

Terminates a given composite model.

\[
\text{status} := \text{oms\_terminate}(\text{cref});
\]
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 OM-Simulator. The independent stand-alone version doesn’t provide any graphical modelling editor.

See also FMI documentation.

![OMEdit Main Window and Browsers](image)

Fig. 1: OMEdit MainWindow and Browsers.

### 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

Chapter 7. Graphical Modelling
7.2 Add System

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

![OMEdit: Add System](image)

Fig. 4: OMEdit: Add System

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

![OMEdit - Add System](image)

Fig. 5: OMEdit: Add System Dialog

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:

7.2. Add System
Fig. 6: OMEdit: Add SubModel
That will pop-up a dialog to enter the names of the new sub-model.

![OMEdit: Add SubModel Dialog](image)

**Fig. 7: OMEdit: Add SubModel Dialog**

### 7.4 Instantiate Model

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

![OMEdit: Instantiate Model](image)

**Fig. 8: OMEdit: Instantiate Model**

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

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

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

8.1 Examples

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

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

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

The goal is to estimate the value of the coefficient $a$ and the initial value $x_{\text{start}}$ of the state variable $x$. Instead of real measurements, the script simply uses simulation data generated from the HelloWorld examples as measurement data. The array `data_time` contains the time instants at which a sample is taken and the array `data_x` contains the value of $x$ that corresponds to the respective time instant.

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

Listing 1: HelloWorld_cs_Fit.lua

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

-- add FMU
oms_addSubModel("HelloWorld_cs_Fit.root.HelloWorld", ".../FMUs/HelloWorld.fmufmu")
```
-- create system identification model for model
simodel = omsi_newSysIdentModel("HelloWorld_cs_Fit");

-- Data generated from simulating HelloWorld.mo for 1.0s with Euler and 0.1s step size
kNumSeries = 1;
kNumObservations = 11;
data_time = {0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1};
inputvars = {};
measurementvars = {"HelloWorld_cs_Fit.root.HelloWorld.x"};
data_x = {1, 0.9, 0.8100000000000001, 0.7290000000000001, 0.6561, 0.5904900000000001, 0.5314410000000001, 0.4782969000000001, 0.43046721, 0.387420489, 0.3486784401};

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

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

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

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

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

omsi_freeSysIdentModel(simodel)

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

Running the script generates the following console output:

iter | cost | cost_change | |gradient| |step| | tr_ratio| | tr。
|radius | ls_iter | iter_time | total_time|
|0 | 4.034320e-01 | 0.00e+00 | 2.19e+00 | 0.00e+00 | 0.00e+00 | 1.00e+04 |
|1 | 2.35e-02 | 2.35e-02 |
|1 | 3.821520e-02 | 3.65e-01 | 4.11e-01 | 9.87e-01 | 9.06e-01 | 2.15e+04 |
|1 | 2.90e-02 | 5.25e-02 |
OMSimulator Documentation, Release v2.1.0-dev-162-g63d0b36

2  6.837776e-04  3.75e-02  5.19e-02  3.58e-01  9.83e-01  6.46e+04
←  1  2.77e-02  8.02e-02
3  1.354499e-07  6.84e-04  6.08e-04  4.58e-02  1.00e+00  1.94e+05
←  1  2.96e-02  1.10e-01
4  5.854620e-15  1.35e-07  1.09e-07  7.22e-04  1.00e+00  5.82e+05
←  1  3.08e-02  1.41e-01
5  1.160287e-25  5.85e-15  2.26e-13  1.59e-07  1.00e+00  1.74e+06
←  1  2.86e-02  1.69e-01

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

HelloWorld_cs_Fit.HelloWorld:a(start=-0.5, *estimate*=-1.0480741793778)
HelloWorld_cs_Fit.HelloWorld:x_start(start=0.5, *estimate*=1)

8.2 Lua Scripting Commands

8.2.1 omsi_addInput

Add input values for external model inputs.

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

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

8.2.2 omsi_addMeasurement

Add measurement values for a fitting variable.

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

8.2. Lua Scripting Commands
8.2.3 omsi_addParameter

Add parameter that should be estimated.

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

8.2.4 omsi_describe

Print summary of SysIdent model.

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

8.2.5 omsi_freeSysIdentModel

Unloads a model.

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

8.2.6 omsi_getParameter

Get parameter that should be estimated.

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

8.2.7 omsi_getState

Get state of SysIdent model object.

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

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

```lua
-- simodel [inout] SysIdent model as opaque pointer.
-- nSeries [in] Number of measurement series.
-- time [in] Array of measurement/input time instants.
-- measurmentvars [in] Array of names of observed measurement variables.
-- status [out] Error status.
status = omsi_initialize(simodel, nSeries, time, inputvars, measurmentvars)
```

8.2.9 omsi_newSysIdentModel

Creates an empty model for parameter estimation.

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

8.2.10 omsi_setOptions_max_num_iterations

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

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

8.2.11 omsi_solve

Solve parameter estimation problem.

```lua
-- simodel [inout] SysIdent model as opaque pointer.
-- reporttype [in] Print report and progress information after call to Ceres solver.
-- status [out] Error status.
status = omsi_solve(simodel, reporttype)
```
Composite models are imported and exported in the *System Structure Description (SSD)* format, which is part of the *System Structure and Parameterization (SSP)* standard.

## 9.1 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:Annotations/>
      <oms:Bus name="bus2">
        <oms:Signals>
          <oms:Signal name="C1"/>
          <oms:Signal name="C2"/>
        </oms:Signals>
      </oms:Bus>
    </ssd:System>
    <ssd:System name="WC1">
      <ssd:Connectors/>
      <ssd:Annotations/>
      <oms:Bus name="bus1">
        <oms:Signals>
          <oms:Signal name="C1"/>
          <oms:Signal name="C2"/>
          <oms:Signal name="C3"/>
        </oms:Signals>
      </oms:Bus>
    </ssd:System>
  </ssd:Elements>
</ssd:SystemStructureDescription>
```

(continues on next page)
9.2 TLM Systems

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

```xml
<?xml version="1.0"?>
<ssd:System name="tlm">
  <ssd:SimulationInformation>
    <ssd:Annotations>
      <ssd:Annotation type="org.openmodelica">
        <oms:TlmMaster ip="127.0.1.1" managerport="11111" monitorport="11121"/>
      </ssd:Annotation>
    </ssd:Annotations>
  </ssd:SimulationInformation>
  <ssd:Elements>
    <ssd:System name="weaklycoupled">
      <ssd:SimulationInformation>
        <!-- (continues on next page) -->
      </ssd:SimulationInformation>
    </ssd:System>
  </ssd:Elements>
</ssd:System>
```

(continues on next page)
9.3 TLM Connections

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

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

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;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>

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>
</tbody>
</table>

Continued on next 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, 73
OMSimulator, 1
  Examples, 4
  Flags, 3
OMSimulatorLib, 4
  C-API, 5
OMSimulatorLua, 20
  Examples, 21
  Scripting Commands, 21
OMSimulatorPython, 37
  Examples, 39
  Scripting Commands, 39
OMSysIdent, 80
  Examples, 81
  Scripting Commands, 83
OpenModelicaScripting, 55
  Examples, 57
  Scripting Commands, 57

S

SSP, 85
  Bus connections, 87
  TLM connections, 89
  TLM Systems, 88