# CONTENTS

1 Introduction 1

2 OMSimulator 3
   2.1 OMSimulator Flags 3
   2.2 Examples 4

3 OMSimulatorLib 5
   3.1 C-API 5
      3.1.1 RunFile 5
      3.1.2 addBus 5
      3.1.3 addConnection 5
      3.1.4 addConnector 5
      3.1.5 addConnectorToBus 6
      3.1.6 addConnectorToTLMBus 6
      3.1.7 addExternalModel 6
      3.1.8 addResources 6
      3.1.9 addSignalsToResults 6
      3.1.10 addSubModel 6
      3.1.11 addSystem 6
      3.1.12 addTLMBus 7
      3.1.13 addTLMConnection 7
      3.1.14 compareSimulationResults 7
      3.1.15 copySystem 7
      3.1.16 delete 8
      3.1.17 deleteConnection 8
      3.1.18 deleteConnectorFromBus 8
      3.1.19 deleteConnectorFromTLMBus 8
      3.1.20 deleteResources 8
      3.1.21 doStep 9
      3.1.22 export 9
      3.1.23 exportDependencyGraphs 9
      3.1.24 exportSSMTemplate 9
      3.1.25 exportSSVTemplate 9
      3.1.26 exportSnapshot 10
      3.1.27 extractFMKind 10
      3.1.28 faultInjection 10
      3.1.29 freeMemory 10
      3.1.30 getBoolean 10
      3.1.31 getBus 10
3.1.32 getComponentType ................................. 11
3.1.33 getConnections .................................. 11
3.1.34 getConnector ...................................... 11
3.1.35 getDirectionalDerivative ......................... 11
3.1.36 getElement ........................................ 11
3.1.37 getElements ....................................... 11
3.1.38 getFMUInfo ....................................... 11
3.1.39 getFixedStepSize ................................ 12
3.1.40 getInteger ....................................... 12
3.1.41 getModelState .................................... 12
3.1.42 getReal .......................................... 12
3.1.43 getResultFile ..................................... 12
3.1.44 getSolver ......................................... 12
3.1.45 getStartTime ..................................... 12
3.1.46 getStopTime ...................................... 12
3.1.47 getString ......................................... 13
3.1.48 getSubModelPath ................................. 13
3.1.49 getSystemType .................................... 13
3.1.50 getTLMBus ........................................ 13
3.1.51 getTLMVariableTypes ......................... 13
3.1.52 getTime .......................................... 13
3.1.53 getTolerance ..................................... 14
3.1.54 getVariableStepSize ............................. 14
3.1.55 getVersion ........................................ 14
3.1.56 importFile ....................................... 14
3.1.57 importSnapshot ................................... 14
3.1.58 initialize ......................................... 14
3.1.59 instantiate ....................................... 15
3.1.60 list ................................................. 15
3.1.61 listUnconnectedConnectors ..................... 15
3.1.62 loadSnapshot ..................................... 15
3.1.63 newModel ......................................... 15
3.1.64 newResources ..................................... 15
3.1.65 referenceResources .............................. 16
3.1.66 removeSignalsFromResults ..................... 16
3.1.67 rename ............................................ 16
3.1.68 reset ............................................... 16
3.1.69 setActivationRatio ................................ 17
3.1.70 setBoolean ....................................... 17
3.1.71 setBusGeometry ................................... 17
3.1.72 setCommandLineOption ........................... 17
3.1.73 setConnectionGeometry ......................... 19
3.1.74 setConnectorGeometry ............................ 19
3.1.75 setElementGeometry ................................ 19
3.1.76 setFixedStepSize ................................ 19
3.1.77 setInteger ........................................ 19
3.1.78 setLogFile ....................................... 19
3.1.79 setLoggingCallback .............................. 19
3.1.80 setLoggingInterval .............................. 20
3.1.81 setLoggingLevel .................................. 20
3.1.82 setMaxLogFileSize .............................. 20
3.1.83 setReal ................................................................. 20
3.1.84 setRealInputDerivative ........................................ 20
3.1.85 setResultFile ......................................................... 21
3.1.86 setSolver .............................................................. 21
3.1.87 setStartTime ........................................................ 21
3.1.88 setStopTime ......................................................... 21
3.1.89 setString ............................................................ 21
3.1.90 setTLMBusGeometry ............................................... 22
3.1.91 setTLMConnectionParameters ............................... 21
3.1.92 setTLMPositionAndOrientation .................................. 22
3.1.93 setTLMSocketData ............................................... 22
3.1.94 setTempDirectory ............................................... 22
3.1.95 setTolerance ....................................................... 22
3.1.96 setVariableStepSize ............................................ 22
3.1.97 setWorkingDirectory ............................................ 23
3.1.98 simulate ............................................................. 23
3.1.99 simulate_realtime ............................................... 23
3.1.100 stepUntil .......................................................... 23
3.1.101 terminate ........................................................ 23

4 OMSimulatorLua .......................................................... 25
4.1 Examples ............................................................... 25
4.2 Lua Scripting Commands ............................................ 25
4.2.1 addBus .............................................................. 25
4.2.2 addConnection ..................................................... 26
4.2.3 addConnector ...................................................... 26
4.2.4 addConnectorToBus ............................................... 26
4.2.5 addConnectorToTLMBus ........................................... 26
4.2.6 addExternalModel ................................................ 26
4.2.7 addResources ...................................................... 27
4.2.8 addSignalsToResults ............................................. 27
4.2.9 addSubModel ........................................................ 27
4.2.10 addSystem .......................................................... 27
4.2.11 addTLMBus ........................................................ 27
4.2.12 addTLMConnection .............................................. 28
4.2.13 compareSimulationResults .................................... 28
4.2.14 copySystem ........................................................ 28
4.2.15 delete ............................................................... 28
4.2.16 deleteConnection .................................................. 29
4.2.17 deleteConnectorFromBus ....................................... 29
4.2.18 deleteConnectorFromTLMBus .................................. 29
4.2.19 deleteResources ................................................... 29
4.2.20 export ............................................................... 30
4.2.21 exportDependencyGraphs ....................................... 30
4.2.22 exportSSMTemplate .............................................. 30
4.2.23 exportSSVTemplate .............................................. 30
4.2.24 exportSnapshot .................................................... 30
4.2.25 faultInjection ...................................................... 31
4.2.26 freeMemory ........................................................ 31
4.2.27 getBoolean ........................................................ 31
4.2.28 getDirectionalDerivative ....................................... 31
5 OMSimulatorPython

5.1 Examples ................................................. 43
5.2 Python Scripting Commands ......................... 44

5.2.1 addBus ........................................... 44
5.2.2 addConnection ..................................... 44
5.2.3 addConnector ...................................... 44
5.2.4 addConnectorToBus .................................. 45
5.2.5 addConnectorToTLMBus ............................. 45
5.2.6 addExternalModel ................................... 45
5.2.7 addResources ...................................... 45
5.2.8 addSignalsToResults ............................... 46
5.2.9 addSubModel ....................................... 46
5.2.10 addSystem ........................................ 46
5.2.11 addTLMBus ........................................ 46
5.2.12 addTLMConnection ................................. 47
5.2.13 compareSimulationResults ....................... 47
5.2.14 copySystem ....................................... 47
5.2.15 delete .............................................. 47
5.2.16 deleteConnection ................................. 47
5.2.17 deleteConnectorFromBus ......................... 48
5.2.18 deleteConnectorFromTLMBus ..................... 48
5.2.19 deleteResources ................................... 48
5.2.20 doStep ............................................. 48
5.2.21 export ............................................. 49
5.2.22 exportDependencyGraphs ......................... 49
5.2.23 exportSSMTemplate ............................... 49
5.2.24 exportSSVTemplate ............................... 49
5.2.25 exportSnapshot ................................... 49
5.2.26 faultInjection ..................................... 50
5.2.27 freeMemory ....................................... 50
5.2.28 getBoolean ....................................... 50
5.2.29 getDirectionalDerivative ......................... 50
5.2.30 getFixedStepSize .................................. 50
5.2.31 getInteger ........................................ 50
5.2.32 getReal .......................................... 51
5.2.33 getResultFile ...................................... 51
5.2.34 getSolver .......................................... 51
5.2.35 getStart_time ..................................... 51
5.2.36 getStopTime ...................................... 51
5.2.37 getString .......................................... 51
5.2.38 getSubModelPath .................................. 51
5.2.39 getSystemType .................................... 51
5.2.40 getTime ........................................... 52
5.2.41 getTolerance ...................................... 52
5.2.42 getVariableStepSize ............................... 52
5.2.43 getVersion ......................................... 52
5.2.44 importFile ....................................... 52
5.2.45 importSnapshot .................................... 52
5.2.46 initialize ......................................... 53
5.2.47 instantiate .............................................. 53
5.2.48 list ...................................................... 53
5.2.49 listUnconnectedConnectors ......................... 53
5.2.50 loadSnapshot ......................................... 53
5.2.51 newModel ............................................. 53
5.2.52 newResources ......................................... 54
5.2.53 referenceResources ................................. 54
5.2.54 removeSignalsFromResults ........................... 55
5.2.55 rename ................................................ 55
5.2.56 reset .................................................. 55
5.2.57 setBoolean .......................................... 55
5.2.58 setCommandLineOption .............................. 55
5.2.59 setFixedStepSize .................................... 57
5.2.60 setInteger .......................................... 57
5.2.61 setLogFile .......................................... 57
5.2.62 setLoggingInterval .................................. 57
5.2.63 setLoggingLevel ..................................... 57
5.2.64 setMaxLogFileSize .................................. 58
5.2.65 setReal ............................................. 58
5.2.66 setRealInputDerivative ............................. 58
5.2.67 setResultFile ........................................ 58
5.2.68 setSolver .......................................... 58
5.2.69 setStartTime ........................................ 59
5.2.70 setStopTime ......................................... 59
5.2.71 setString ........................................... 59
5.2.72 setTempDirectory .................................... 59
5.2.73 setTolerance ........................................ 59
5.2.74 setVariableStepSize ................................ 60
5.2.75 setWorkingDirectory ............................... 60
5.2.76 simulate ............................................ 60
5.2.77 stepUntil ............................................ 60
5.2.78 terminate .......................................... 60

5.3 Example: Pi .............................................. 60

6 OpenModelicaScripting .................................... 63
6.1 Examples ................................................... 63
6.2 OpenModelica Scripting Commands .................... 63
  6.2.1 addBus ............................................. 63
  6.2.2 addConnection ....................................... 64
  6.2.3 addConnector ....................................... 64
  6.2.4 addConnectorToBus .................................. 64
  6.2.5 addConnectorToTLMBus .............................. 64
  6.2.6 addExternalModel .................................. 64
  6.2.7 addSignalsToResults ............................... 65
  6.2.8 addSubModel ........................................ 65
  6.2.9 addSystem .......................................... 65
  6.2.10 addTLMBus ......................................... 65
  6.2.11 addTLMConnection ................................ 66
  6.2.12 compareSimulationResults ........................ 66
  6.2.13 copySystem ........................................ 66
  6.2.14 delete ............................................. 66
6.2.15 deleteConnection .................................................. 66
6.2.16 deleteConnectorFromBus ........................................... 67
6.2.17 deleteConnectorFromTLMBus ..................................... 67
6.2.18 export ............................................................... 67
6.2.19 exportDependencyGraphs ......................................... 67
6.2.20 exportSnapshot .................................................... 67
6.2.21 extractFMIKind ..................................................... 67
6.2.22 faultInjection ...................................................... 67
6.2.23 freeMemory ......................................................... 68
6.2.24 getBoolean ......................................................... 68
6.2.25 getFixedStepSize .................................................. 68
6.2.26 getInteger .......................................................... 68
6.2.27 getModelState ...................................................... 68
6.2.28 getReal .............................................................. 69
6.2.29 getSolver ............................................................ 69
6.2.30 getStartTime ....................................................... 69
6.2.31 getStopTime ......................................................... 69
6.2.32 getSubModelPath .................................................. 69
6.2.33 getSystemType ..................................................... 69
6.2.34 getTime ............................................................. 69
6.2.35 getTolerance ...................................................... 70
6.2.36 getVariableStepSize ............................................. 70
6.2.37 getVersion .......................................................... 70
6.2.38 importFile .......................................................... 70
6.2.39 importSnapshot ................................................... 70
6.2.40 initialize ........................................................... 70
6.2.41 instantiate ......................................................... 70
6.2.42 list ................................................................. 71
6.2.43 listUnconnectedConnectors ....................................... 71
6.2.44 loadSnapshot ..................................................... 71
6.2.45 newModel ........................................................... 71
6.2.46 removeSignalsFromResults ....................................... 71
6.2.47 rename ............................................................... 71
6.2.48 reset ................................................................. 72
6.2.49 setBoolean .......................................................... 72
6.2.50 setCommandLineOption ........................................... 72
6.2.51 setFixedStepSize .................................................. 73
6.2.52 setInteger ........................................................... 73
6.2.53 setLogFile .......................................................... 74
6.2.54 setLoggingInterval ............................................... 74
6.2.55 setLoggingLevel ................................................... 74
6.2.56 setReal ............................................................... 74
6.2.57 setRealInputDerivative .......................................... 74
6.2.58 setResultFile ....................................................... 75
6.2.59 setSolver ............................................................ 75
6.2.60 setStartTime ....................................................... 75
6.2.61 setStopTime ......................................................... 75
6.2.62 setTLMPositionAndOrientation ................................ 75
6.2.63 setTLMSocketData ................................................ 76
6.2.64 setTempDirectory ................................................ 76
6.2.65 setTolerance ........................................................ 76
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`:

```
Usage: OMSimulator [Options] [Lua script] [FMU] [SSP file]
Options:
  --addParametersToCSV=<arg>  Export parameters to .csv file
  (true, [false])
  --algLoopSolver=<arg>  Specifies the alg. loop solver
  method (fixedpoint, [kinsol]) used for algebraic loops spanning over
  multiple components.
  --clearAllOptions  Reset all flags to default
  --deleteTempFiles=<bool>  Deletes temp files as soon as
  they are no longer needed ([true], false)
  --directionalDerivatives=<bool>  Specifies whether directional
  derivatives should be used to calculate the Jacobian for alg. loops or
  if a numerical approximation should be used instead ([true], false)
  --dumpAlgLoops=<bool>  Dump information for alg loops
  (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
  --ignoreInitialUnknowns=<bool>  Ignore the initial unknowns
  --inputExtrapolation=<bool>  Enables input extrapolation
  --intervals=<int>  Specifies the number of
  communication points (arg > 1)
  --logFile=<arg>  Specifies the logfile (stdout is used
  if no log file is specified)
  --logLevel=<int>  0 default, 1 debug, 2
  --maxEventIteration=<int>  Specifies the max. number of
  iterations for handling a single event
  --maxLoopIteration=<int>  Specifies the max. number of
  iterations for solving algebraic loops between system-level components
  Internal algebraic loops of components are not affected.
```

(continues on next page)
OMSimulator Documentation, Release v2.1.1.post155-gc803efc

(continued from previous page)

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--mode=&lt;arg&gt;</td>
<td>Forces a certain FMI mode if the FMU provides cs and me (cs, [me])</td>
</tr>
<tr>
<td>--numProcs=&lt;int&gt;</td>
<td>Specifies the max. number of processors to use (0=auto, 1=default)</td>
</tr>
<tr>
<td>--progressBar=&lt;bool&gt;</td>
<td>Shows a progress bar for the simulation progress in the terminal (true, [false])</td>
</tr>
<tr>
<td>--realTime=&lt;bool&gt;</td>
<td>Experimental feature for (soft) real-time co-simulation (true, [false])</td>
</tr>
<tr>
<td>--resultFile=&lt;arg&gt;</td>
<td>Specifies the name of the output result file</td>
</tr>
<tr>
<td>--skipCSVHeader=&lt;bool&gt;</td>
<td>Skip exporting the scv delimiter in the header ([true], false)</td>
</tr>
<tr>
<td>--solver=&lt;arg&gt;</td>
<td>Specifies the integration method (euler, [cvode])</td>
</tr>
<tr>
<td>--solverStats=&lt;bool&gt;</td>
<td>Adds solver stats to the result file, e.g. step size; not supported for all solvers (true, [false])</td>
</tr>
<tr>
<td>--startTime=&lt;double&gt;</td>
<td>Specifies the start time</td>
</tr>
<tr>
<td>--stepSize=&lt;arg&gt;</td>
<td>Specifies the step size (&lt;step size&gt; or &lt;init step,min step,max step&gt;)</td>
</tr>
<tr>
<td>--stopTime=&lt;double&gt;</td>
<td>Specifies the stop time</td>
</tr>
<tr>
<td>--stripRoot=&lt;bool&gt;</td>
<td>Removes the root system prefix from all exported signals (true, [false])</td>
</tr>
<tr>
<td>--suppressPath=&lt;bool&gt;</td>
<td>Suppresses path information in info messages; especially useful for testing ([true], false)</td>
</tr>
<tr>
<td>--tempDir=&lt;arg&gt;</td>
<td>Specifies the temp directory</td>
</tr>
<tr>
<td>--timeout=&lt;int&gt;</td>
<td>Specifies the maximum allowed time in seconds for running a simulation (0 disables)</td>
</tr>
<tr>
<td>--tolerance=&lt;double&gt;</td>
<td>Specifies the relative tolerance</td>
</tr>
<tr>
<td>--version [-v]</td>
<td>Displays version information</td>
</tr>
<tr>
<td>--wallTime=&lt;bool&gt;</td>
<td>Add wall time information for the result file</td>
</tr>
<tr>
<td>--workingDir=&lt;arg&gt;</td>
<td>Specifies the working directory</td>
</tr>
<tr>
<td>--zeroNominal=&lt;bool&gt;</td>
<td>Using this flag, FMUs with invalid nominal values will be accepted and the invalid nominal values will be replaced with 1.0</td>
</tr>
</tbody>
</table>

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

### 2.2 Examples

```bash
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 addResources

Adds an external resources to an existing SSP. The external resources should be a “.ssv” or “.ssm” file.

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

3.1.9 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.

3.1.10 addSubModel

Adds a component to a system.

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

3.1.11 addSystem

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

Adds a TLM bus.

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

3.1.13 addTLMConnection

Connects two TLM connectors.

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

3.1.14 compareSimulationResults

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

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

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.

```
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 deleteResources

Deletes the reference and resource file in a SSP. Deletion of “.ssv” and “.ssm” files are currently supported. The API can be used in two ways.

1. deleting only the reference file in “.ssd”.
2. deleting both reference and resource files in “.ssp”.

To delete only the reference file in ssd, the user should provide the full qualified cref of the “.ssv” file associated with a system or subsystem or component (e.g) “model.root:root1.ssv”.

To delete both the reference and resource file in ssp, it is enough to provide only the model cref of the “.ssv” file (e.g) “model:root1.ssv”.

When deleting only the references of a “.ssv” file, if a parameter mapping file “.ssm” is binded to a “.ssv” file then the “.ssm” file will also be deleted. It is not possible to delete the references of “.ssm” seperately as the ssm file is binded to a ssv file.

The filename of the reference or resource file is provided by the users using colon suffix at the end of cref. (e.g) “:root.ssv”
3.1.21 **doStep**

Simulates a macro step of the given composite model. The step size will be determined by the master algorithm and is limited by the defined minimal and maximal step sizes.

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

3.1.22 **export**

Exports a composite model to a SPP file.

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

3.1.23 **exportDependencyGraphs**

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

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

3.1.24 **exportSSMTemplate**

Exports all signals that have start values of one or multiple FMUs to a SSM file that are read from modelDescription.xml with a mapping entry. The mapping entry specifies a single mapping between a parameter in the source and a parameter of the system or component being parameterized. The mapping entry contains two attributes namely source and target. The source attribute will be empty and needs to be manually mapped by the users associated with the parameter name defined in the SSV file, the target contains the name of parameter in the system or component to be parameterized. The function can be called for a top level model or a certain FMU component. If called for a top level model, start values of all FMUs are exported to the SSM file. If called for a component, start values of just this FMU are exported to the SSM file.

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

3.1.25 **exportSSVTemplate**

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

```c
oms_status_enu_t oms_exportSSVTemplate(const char* cref, const char* filename);
```
3.1.26 exportSnapshot

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

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

3.1.27 extractFMIKind

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

```c
oms_status_enu_t oms_extractFMIKind(const char* filename, oms_fmi_kind_enu_t* kind);
```

3.1.28 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.$original + \text{faultValue} )</td>
</tr>
<tr>
<td>oms_fault_type_gain</td>
<td>( y = y.$original \times \text{faultValue} )</td>
</tr>
<tr>
<td>oms_fault_type_const</td>
<td>( y = \text{faultValue} )</td>
</tr>
</tbody>
</table>

3.1.29 freeMemory

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

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

3.1.30 getBoolean

Get boolean value of given signal.

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

3.1.31 getBus

Gets the bus object.

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

Gets the type of the given component.

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

3.1.33 getConnections

Get list of all connections from a given component.

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

3.1.34 getConnector

Gets the connector object of the given connector cref.

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

3.1.35 getDirectionalDerivative

This function computes the directional derivatives of an FMU.

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

3.1.36 getElement

Get element information of a given component reference.

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

3.1.37 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);
```

3.1.38 getFMUInfo

Returns FMU specific information.

```c
oms_status_enu_t oms_getFMUInfo(const char* cref, const oms_fmu_info_t** fmuInfo);
```
3.1.39 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);
```

3.1.40 getInteger

Get integer value of given signal.

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

3.1.41 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);
```

3.1.42 getReal

Get real value.

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

3.1.43 getResultFile

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

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

3.1.44 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);
```

3.1.45 getStartTime

Get the start time from the model.

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

Get the stop time from the model.

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

3.1.47 getString

Get string value.

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

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

3.1.48 getSubModelPath

Returns the path of a given component.

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

3.1.49 getSystemType

Gets the type of the given system.

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

3.1.50 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);
```

3.1.51 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);
```

3.1.52 getTime

Get the current simulation time from the model.
3.1.53 getTolerance

Gets the tolerance of a given system or component.

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

3.1.54 getVariableStepSize

Gets the step size parameters.

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

3.1.55 getVersion

Returns the library’s version string.

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

3.1.56 importFile

Imports a composite model from a SSP file.

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

3.1.57 importSnapshot

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

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

3.1.58 initialize

Initializes a composite model.

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

Instantiates a given composite model.

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

3.1.60 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);
```

3.1.61 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);
```

3.1.62 loadSnapshot

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

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

3.1.63 newModel

Creates a new and yet empty composite model.

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

3.1.64 newResources

Adds a new empty resources to the SSP. The resource file is a “.ssv” file where the parameter values set by the users using “oms_setReal()”, “oms_setInteger()” and “oms_setReal()” are written to the file. Currently only “.ssv” files can be created.

The filename of the resource file is provided by the users using colon suffix at the end of `cref`. (e.g) “:root.ssv”
3.1.65 referenceResources

Switches the references of “.ssv” and “.ssm” in a SSP file. Referencing of “.ssv” and “.ssm” files are currently supported. The API can be used in two ways.

1. Referencing only the “.ssv” file.
2. Referencing both the “.ssv” along with the “.ssm” file.

This API should be used in combination with “oms_deleteResources”. To switch with a new reference, the old reference must be deleted first using “oms_deleteResources” and then reference with new resources.

When deleting only the references of a “.ssv” file, if a parameter mapping file “.ssm” is binded to a “.ssv” file, then the reference of “.ssm” file will also be deleted. It is not possible to delete the references of “.ssm” seperately as the ssm file is binded to a ssv file. Hence it is not possible to switch the reference of “.ssm” file alone. So inorder to switch the reference of “.ssm” file, the users need to bind the reference of “.ssm” file along with the “.ssv”.

The filename of the reference or resource file is provided by the users using colon suffix at the end of cref (e.g) “:root.ssv”, and the “.ssm” file is optional and is provided by the user as the second argument to the API.

3.1.66 removeSignalsFromResults

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

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

3.1.67 rename

Renames a model, system, or component.

3.1.68 reset

Reset the composite model after a simulation run.

The FMUs go into the same state as after instantiation.
3.1.69 `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);
```

3.1.70 `setBoolean`

Sets the value of a given boolean signal.

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

3.1.71 `setBusGeometry`

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

3.1.72 `setCommandLineOption`

Sets special flags.

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

Available flags:

- **info:** Usage: OMSimulator [Options] [Lua script] [FMU] [SSP file]
  
  Options:
  ```
  --addParametersToCSV=<arg> Export parameters to .csv file
  ```
  ```
  --algLoopSolver=<arg> Specifies the alg. loop solver
  ```
  ```
  method (fixedpoint, [kinsol]) used for algebraic loops spanning over
  ```
  ```
  multiple components.
  ```
  --clearAllOptions Reset all flags to default
  ```
  ```
  --deleteTempFiles=<bool> Deletes temp files as soon as
  ```
  ```
  they are no longer needed ((true), [false])
  ```
  ```
  --directionalDerivatives=<bool> Specifies whether directional
  ```
  ```
  derivatives should be used to calculate the Jacobian for alg. loops or
  ```
  ```
  if a numerical approximation should be used instead ((true), [false])
  ```
  ```
  --dumpAlgLoops=<bool> Dump information for alg loops
  ```
  ```
  --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
  ```
  ```
  ```
  (continues on next page)
--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)
--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
--maxLoopIteration=<int> Specifies the max. number of iterations for solving algebraic loops between system-level components. Internal algebraic loops of components are not affected.
--mode=<arg> [-m] Forces a certain FMI mode iff 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
--skipCSVHeader=<arg> Skip exporting the scv delimiter in the header ([true], false)
--solver=<arg> Specifies the integration method (euler, [cvode])
--solverStats=<bool> Adds solver stats to the result file, e.g. step size; not supported for all solvers (true, [false])
--startTime=<double> Specifies the start time
--stepSize=<arg> Specifies the step size (<step size> or <init step,min step,max step>)
--stopTime=<double> 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)
--version [-v] Displays version information
--wallTime=<bool> Add wall time information for the result file (true, [false])
--workingDir=<arg> Specifies the working directory
--zeroNominal=<bool> Using this flag, FMUs with invalid nominal values will be accepted and the invalid nominal values will be replaced with 1.0
3.1.73 setConnectionGeometry

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

3.1.74 setConnectorGeometry

Set geometry information to a given connector.

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

3.1.75 setElementGeometry

Set geometry information to a given component.

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

3.1.76 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);
```

3.1.77 setInteger

Sets the value of a given integer signal.

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

3.1.78 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);
```

3.1.79 setLoggingCallback

Sets a callback function for the logging system.

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

Set the logging interval of the simulation.

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

3.1.81 **setLoggingLevel**

Enables/Disables debug logging (logDebug and logTrace).

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

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

3.1.82 **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);
```

3.1.83 **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.

3.1.84 **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);
```
3.1.85 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.

3.1.86 setSolver

Sets the solver method for the given system.

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

3.1.87 setStartTime

Set the start time of the simulation.

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

3.1.88 setStopTime

Set the stop time of the simulation.

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

3.1.89 setString

Sets the value of a given string signal.

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

3.1.90 setTLMBusGeometry

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

3.1.91 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);
```
3.1.92 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);
```

3.1.93 setTLMSocketData

Sets data for TLM socket communication.

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

3.1.94 setTempDirectory

Set new temp directory.

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

3.1.95 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.

3.1.96 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);
```
3.1.97 setWorkingDirectory

Set a new working directory.

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

3.1.98 simulate

Simulates a composite model.

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

3.1.99 simulate_realtime

Experimental feature for (soft) real-time simulation.

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

3.1.100 stepUntil

Simulates a composite model until a given time value.

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

3.1.101 terminate

Terminates a given composite model.

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

Adds a bus to a given component.

```lua
status = oms_addBus(cref)
```
4.2.2 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.

4.2.3 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

4.2.4 addConnectorToBus

Adds a connector to a bus.

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

4.2.5 addConnectorToTLMBus

Adds a connector to a TLM bus.

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

4.2.6 addExternalModel

Adds an external model to a TLM system.

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

Adds an external resources to an existing SSP. The external resources should be a “.ssv” or “.ssm” file.

```lua
status = oms_addResources(cref, path)

-- Example
oms_importFile("addExternalResources1.ssp")
-- add list of external resources from filesystem to ssp
oms_addResources("addExternalResources", "../../resources/externalRoot.ssv")
oms_addResources("addExternalResources:externalSystem.ssv", "../../resources/externalSystem1.ssv")
oms_addResources("addExternalResources", "../../resources/externalGain.ssv")
-- export the ssp with new resources
oms_export("addExternalResources", "addExternalResources1.ssp")
```

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
```

(continues on next page)
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.

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

### 4.2.13 compareSimulationResults

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

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

The following table describes the input values:

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

The following table describes the return values:

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

### 4.2.14 copySystem

Copies a system.

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

### 4.2.15 delete

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

Deletes the connection between connectors $cref_A$ and $cref_B$.

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

The two arguments $cref_A$ and $cref_B$ get swapped automatically if necessary.

4.2.17 deleteConnectorFromBus

Deletes a connector from a given bus.

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

4.2.18 deleteConnectorFromTLMBus

Deletes a connector from a given TLM bus.

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

4.2.19 deleteResources

Deletes the reference and resource file in a SSP. Deletion of “.ssv” and “.ssm” files are currently supported. The API can be used in two ways.

1. deleting only the reference file in “.ssd”.
2. deleting both reference and resource files in “.ssp”.

To delete only the reference file in ssd, the user should provide the full qualified cref of the “.ssv” file associated with a system or subsystem or component (e.g) “model.root:root1.ssv”.

To delete both the reference and resource file in ssp, it is enough to provide only the model cref of the “.ssv” file (e.g) “model:root1.ssv”.

When deleting only the references of a “.ssv” file, if a parameter mapping file “.ssm” is binded to a “.ssv” file then the “.ssm” file will also be deleted. It is not possible to delete the references of “.ssm” seperately as the ssm file is binded to a ssv file.

The filename of the reference or resource file is provided by the users using colon suffix at the end of cref. (e.g) “:root.ssv”

```lua
status = oms_deleteResources(cref)
```

-- Example
```lua
oms_importFile("deleteResources1.ssp")
oms_deleteResources("deleteResources.root:root1.ssv")
```

(continues on next page)
oms_deleteResources("deleteResources:root.ssv")
oms_export("deleteResources1.ssp")

### 4.2.20 export

Exports a composite model to a SPP file.

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

### 4.2.21 exportDependencyGraphs

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

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

### 4.2.22 exportSSMTemplate

Exports all signals that have start values of one or multiple FMUs to a SSM file that are read from modelDescription.xml with a mapping entry. The mapping entry specifies a single mapping between a parameter in the source and a parameter of the system or component being parameterized. The mapping entry contains two attributes namely source and target. The source attribute will be empty and needs to be manually mapped by the users associated with the parameter name defined in the SSV file, the target contains the name of parameter in the system or component to be parameterized. The function can be called for a top level model or a certain FMU component. If called for a top level model, start values of all FMUs are exported to the SSM file. If called for a component, start values of just this FMU are exported to the SSM file.

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

### 4.2.23 exportSSVTemplate

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

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

### 4.2.24 exportSnapshot

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

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

Defines a new fault injection block.

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

<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>

4.2.26 freeMemory

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

This function is neither needed nor available from the Lua interface.

4.2.27 getBoolean

Get boolean value of given signal.

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

4.2.28 getDirectionalDerivative

This function computes the directional derivatives of an FMU.

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

4.2.29 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.

\[ \text{stepSize, status} = \text{oms_getFixedStepSize}(\text{cref}) \]

4.2.30 getInteger

Get integer value of given signal.

\[ \text{value, status} = \text{oms_getInteger}(\text{cref}) \]
### 4.2.31 getModelState

Gets the model state of the given model cref.

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

### 4.2.32 getReal

Get real value.

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

### 4.2.33 getSolver

Gets the selected solver method of the given system.

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

### 4.2.34 getStartTime

Get the start time from the model.

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

### 4.2.35 getStopTime

Get the stop time from the model.

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

### 4.2.36 getString

Get string value.

Memory is allocated for `value`. 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
value, status = oms_getString(cref)
```

### 4.2.37 getSystemType

Gets the type of the given system.

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

Get the current simulation time from the model.

```
time, status = oms_getTime(cref)
```

4.2.39 getTolerance

Gets the tolerance of a given system or component.

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

4.2.40 getVariableStepSize

Gets the step size parameters.

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

4.2.41 getVersion

Returns the library’s version string.

```
version = oms_getVersion()
```

4.2.42 importFile

Imports a composite model from a SSP file.

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

4.2.43 importSnapshot

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

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

4.2.44 initialize

Initializes a composite model.

```
status = oms_initialize(cref)
```
4.2.45 instantiate

Instantiates a given composite model.

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

4.2.46 list

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

Memory is allocated for \textit{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.

\[
\text{contents}, \text{status} = \text{oms_list}(\text{cref})
\]

4.2.47 listUnconnectedConnectors

Lists all unconnected connectors of a given system.

Memory is allocated for \textit{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.

\[
\text{contents}, \text{status} = \text{oms_listUnconnectedConnectors}(\text{cref})
\]

4.2.48 loadSnapshot

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

\[
\text{newCref}, \text{status} = \text{oms_loadSnapshot}(\text{cref}, \text{snapshot})
\]

4.2.49 newModel

Creates a new and yet empty composite model.

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

4.2.50 newResources

Adds a new empty resources to the SSP. The resource file is a “.ssv” file where the parameter values set by the users using “oms_setReal()”, “oms_setInteger()” and “oms_setReal()” are written to the file. Currently only “.ssv” files can be created.

The filename of the resource file is provided by the users using colon suffix at the end of cref. (e.g) “:root.ssv”
status = oms_newResources(cref)

-- Example
oms_newModel("newResources")

oms_addSystem("newResources.root", oms_system_wc)
oms_addConnector("newResources.root.Input1", oms_causality_input, oms_signal_type_real)
oms_addConnector("newResources.root.Input2", oms_causality_input, oms_signal_type_real)

-- add Top level new resources, the filename is provided using the colon suffix ":root.ssv"
oms_newResources("newResources.root:root.ssv")
oms_setReal("newResources.root.Input1", 10)
-- export the ssp with new resources
oms_export("newResources", "newResources.ssp")

4.2.51 referenceResources

Switches the references of ".ssv" and ".ssm" in a SSP file. Referencing of ".ssv" and ".ssm" files are currently supported. The API can be used in two ways.

1. Referencing only the ".ssv" file.
2. Referencing both the ".ssv" along with the ".ssm" file.

This API should be used in combination with "oms_deleteResources". To switch with a new reference, the old reference must be deleted first using "oms_deleteResources" and then reference with new resources.

When deleting only the references of a ".ssv" file, if a parameter mapping file ".ssm" is binded to a ".ssv" file, then the reference of ".ssm" file will also be deleted. It is not possible to delete the references of ".ssm" separately as the ssm file is binded to a ssv file. Hence it is not possible to switch the reference of ".ssm" file alone. So inorder to switch the reference of ".ssm" file, the users need to bind the reference of ".ssm" file along with the ".ssv".

The filename of the reference or resource file is provided by the users using colon suffix at the end of cref (e.g) ":root.ssv", and the ".ssm" file is optional and is provided by the user as the second argument to the API.

status = oms_referenceResources(cref, ssmFile)

-- Example
oms_importFile("referenceResources1.ssp")
-- delete only the references in ".ssd" file
oms_deleteResources("referenceResources1.root:root.ssv")
-- usage-1 switch with new references, only ssv file
oms_referenceResources("referenceResources1.root:Config1.ssv")
-- usage-2 switch with new references, both ssv and ssm file
oms_referenceResources("referenceResources1.root:Config1.ssv", "Config1.ssm")
oms_export("referenceResources1.ssp")
4.2.52 removeSignalsFromResults

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

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

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

4.2.53 rename

Renames a model, system, or component.

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

4.2.54 reset

Reset the composite model after a simulation run.

The FMUs go into the same state as after instantiation.

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

4.2.55 setActivationRatio

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

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

4.2.56 setBoolean

Sets the value of a given boolean signal.

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

4.2.57 setCommandLineOption

Sets special flags.

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

Available flags:

```
info:  Usage: OMSimulator [Options] [Lua script] [FMU] [SSP file]
Options:
  --addParametersToCSV=<arg>  Export parameters to .csv file
  --algLoopSolver=<arg>  Specifies the alg. loop solver
   method (fixedpoint, [kinsol]) used for algebraic loops spanning over
   multiple components.
```
--clearAllOptions Reset all flags to default

--deleteTempFiles=<bool> Deletes temp files as soon as they are no longer needed ([true], false)

--directionalDerivatives=<bool> Specifies whether directional derivatives should be used to calculate the Jacobian for alg. loops or if a numerical approximation should be used instead ([true], false)

--dumpAlgLoops=<bool> Dump information for alg loops ([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

--inputExtrapolation=<bool> Enables input extrapolation using derivative information ([true], [false])

--intervals=<int> Specifies the number of communication points (arg > 1)

--logFile=<arg> 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

--maxLoopIteration=<int> Specifies the max. number of iterations for solving algebraic loops between system-level components.

--mode=<arg> Forces a certain FMI mode iff the FMU provides cs and me (cs, [me])

--numProcs=<int> 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> Specifies the name of the output result file

--skipCSVHeader=<arg> Skip exporting the scv delimiter in the header ([true], false), ([true], false)

--solver=<arg> Specifies the integration method (euler, [cvode])

--solverStats=<bool> Adds solver stats to the result file, e.g. step size; not supported for all solvers ([true], [false])

--startTime=<double> Specifies the start time

--stepSize=<arg> Specifies the step size (>step size or <init step,min step,max step>)

--stopTime=<double> 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
--zeroNominal=<bool> Using this flag, FMUs with
→ invalid nominal values will be accepted and the invalid nominal values,
→ will be replaced with 1.0

4.2.58 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)
```

4.2.59 setInteger

Sets the value of a given integer signal.

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

4.2.60 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)
```

4.2.61 setLoggingInterval

Set the logging interval of the simulation.

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

4.2.62 setLoggingLevel

Enables/Disables debug logging (logDebug and logTrace).

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

```lUA
oms_setLoggingLevel(logLevel)
```
4.2.63 setMaxLogFileSize

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

```lua
oms_setMaxLogFileSize(size)
```

4.2.64 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.

4.2.65 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)
```

4.2.66 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.

4.2.67 setSolver

Sets the solver method for the given system.

```lua
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>

### 4.2.68 setStartTime

Set the start time of the simulation.

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

### 4.2.69 setStopTime

Set the stop time of the simulation.

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

### 4.2.70 setString

Sets the value of a given string signal.

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

### 4.2.71 setTLMPositionAndOrientation

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

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

### 4.2.72 setTLMSocketData

Sets data for TLM socket communication.

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

### 4.2.73 setTempDirectory

Set new temp directory.

```lua
status = oms_setTempDirectory(newTempDir)
```
4.2.74 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 their 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.

4.2.75 setVariableStepSize

Sets the step size parameters for methods with stepsize control.

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

4.2.76 setWorkingDirectory

Set a new working directory.

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

4.2.77 simulate

Simulates a composite model.

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

4.2.78 simulate_realtime

Experimental feature for (soft) real-time simulation.

```c
status = experimental_simulate_realtime(ident)
```
4.2.79 stepUntil

Simulates a composite model until a given time value.

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

4.2.80 terminate

Terminates a given composite model.

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

```
> pip3 install OMSimulator --upgrade
```

### 5.1 Examples

```python
from OMSimulator import OMSimulator

oms = OMSimulator()
oms.setTempDirectory("./temp/")
oms.newModel("model")
oms.addSystem("model.root", oms.system_sc)

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

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

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

oms.instantiate("model")
oms.setReal("model.root.system1.x_start", 2.5)
oms.initialize("model")
oms.simulate("model")
oms.terminate("model")
oms.delete("model")
```

The python package also provides a more object oriented API. The following example is equivalent to the previous one:

```python
import OMSimulator as oms
```
```python
oms.setTempDirectory('./temp/')
model = oms.newModel("model")
root = model.addSystem('root', oms.Types.System.SC)

# instantiate FMUs
root.addSubModel('system1', 'FMUs/System1.fmu')
root.addSubModel('system2', 'FMUs/System2.fmu')

# add connections
root.addConnection('system1.y', 'system2.u')
root.addConnection('system2.y', 'system1.u')

# simulation settings
model.resultFile = 'results.mat'
model.stopTime = 0.1
model.fixedStepSize = 1e-4

model.instantiate()
model.setReal('root.system1.x_start', 2.5)
# or system.setReal('system1.x_start', 2.5)

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

## 5.2 Python Scripting Commands

### 5.2.1 addBus

Adds a bus to a given component.

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

### 5.2.2 addConnection

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

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

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

### 5.2.3 addConnector

Adds a connector to a given component.

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

The second argument \texttt{"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

5.2.4 addConnectorToBus

Adds a connector to a bus.

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

5.2.5 addConnectorToTLMBus

Adds a connector to a TLM bus.

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

5.2.6 addExternalModel

Adds an external model to a TLM system.

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

5.2.7 addResources

Adds an external resources to an existing SSP. The external resources should be a “.ssv” or “.ssm” file

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

## Example
from OMSimulator import OMSimulator
oms = OMSimulator()
oms.importFile("addExternalResources1.ssp")
## add list of external resources from filesystem to ssp
oms.addResources("addExternalResources", "/../resources/externalRoot.ssv"
oms.addResources("addExternalResources:externalSystem.ssv", "/../
resources/externalSystem1.ssv")
```
## addExternalResources

Add external resources to the system.

```python
oms.addResources("addExternalResources", "/../resources/externalGain.ssv")
## export the ssp with new resources
oms_export("addExternalResources", "addExternalResources1.ssp")
```

### 5.2.8 addSignalsToResults

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

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

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

### 5.2.9 addSubModel

Adds a component to a system.

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

### 5.2.10 addSystem

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

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

### 5.2.11 addTLMBus

Adds a TLM bus.

```python
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.default
- oms.coarsegrained
- oms.finegrained
5.2.12 addTLMConnection

Connects two TLM connectors.

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

5.2.13 compareSimulationResults

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

```python
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>

5.2.14 copySystem

Copies a system.

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

5.2.15 delete

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

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

5.2.16 deleteConnection

Deletes the connection between connectors crefA and crefB.

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

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

Deletes a connector from a given bus.

status = oms.deleteConnectorFromBus(busCref, connectorCref)

5.2.18 deleteConnectorFromTLMBus

Deletes a connector from a given TLM bus.

status = oms.deleteConnectorFromTLMBus(busCref, connectorCref)

5.2.19 deleteResources

Deletes the reference and resource file in a SSP. Deletion of “.ssv” and “.ssm” files are currently supported. The API can be used in two ways.

1. deleting only the reference file in “.ssd”.
2. deleting both reference and resource files in “.ssp”.

To delete only the reference file in ssd, the user should provide the full qualified cref of the “.ssv” file associated with a system or subsystem or component (e.g) “model.root.root1.ssv”.

To delete both the reference and resource file in ssp, it is enough to provide only the model cref of the “.ssv” file (e.g) “model:root1.ssv”.

When deleting only the references of a “.ssv” file, if a parameter mapping file “.ssm” is binded to a “.ssv” file then the “.ssm” file will also be deleted. It is not possible to delete the references of “.ssm” seperately as the ssm file is binded to a ssv file.

The filename of the reference or resource file is provided by the users using colon suffix at the end of cref. (e.g) “:root.ssv”

status = oms.deleteResources(cref))

## Example

```python
from OMSimulator import OMSimulator
oms = OMSimulator()
oms.importFile("deleteResources1.ssp")
## delete only the references in "ssd" file
oms.deleteResources("deleteResources:root:root.ssv")
## delete both references and resources
oms.deleteResources("deleteResources:root.ssv")
oms.export("deleteResources1.ssp")
```

5.2.20 doStep

Simulates a macro step of the given composite model. The step size will be determined by the master algorithm and is limited by the definied minimal and maximal step sizes.

status = oms.doStep(cref)
5.2.21 export

Exports a composite model to a SPP file.

\[
\text{status} = \text{o.ms.export(cref, filename)}
\]

5.2.22 exportDependencyGraphs

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

\[
\text{status} = \text{o.ms.exportDependencyGraphs(cref, initialization, event, simulation)}
\]

5.2.23 exportSSMTemplate

Exports all signals that have start values of one or multiple FMUs to a SSM file that are read from modelDescription.xml with a mapping entry. The mapping entry specifies a single mapping between a parameter in the source and a parameter of the system or component being parameterized. The mapping entry contains two attributes namely source and target. The source attribute will be empty and needs to be manually mapped by the users associated with the parameter name defined in the SSV file, the target contains the name of parameter in the system or component to be parameterized. The function can be called for a top level model or a certain FMU component. If called for a top level model, start values of all FMUs are exported to the SSM file. If called for a component, start values of just this FMU are exported to the SSM file.

\[
\text{status} = \text{o.ms.exportSSMTemplate(cref, filename)}
\]

5.2.24 exportSSVTemplate

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

\[
\text{status} = \text{o.ms.exportSSVTemplate(cref, filename)}
\]

5.2.25 exportSnapshot

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

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

\[
\text{contents, status} = \text{o.ms.exportSnapshot(cref)}
\]
5.2.26 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>

5.2.27 freeMemory

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

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

5.2.28 getBoolean

Get boolean value of given signal.

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

5.2.29 getDirectionalDerivative

This function computes the directional derivatives of an FMU.

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

5.2.30 getFixedStepSize

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

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

5.2.31 getInteger

Get integer value of given signal.

```
value, status = oms.getInteger(cref)
```
5.2.32 getReal
Get real value.

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

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

```python
filename, bufferSize, status = oms.getResultFile(cref)
```

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

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

5.2.35 getStartTime
Get the start time from the model.

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

5.2.36 getStopTime
Get the stop time from the model.

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

5.2.37 getString
Get string value.

Memory is allocated for `value`. 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
value, status = oms.getString(cref)
```

5.2.38 getSubModelPath
Returns the path of a given component.

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

Gets the type of the given system.

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

5.2.40 getTime

Get the current simulation time from the model.

```python
time, status = oms.getTime(cref)
```

5.2.41 getTolerance

Gets the tolerance of a given system or component.

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

5.2.42 getVariableStepSize

Gets the step size parameters.

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

5.2.43 getVersion

Returns the library’s version string.

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

5.2.44 importFile

Imports a composite model from a SSP file.

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

5.2.45 importSnapshot

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

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

Initializes a composite model.

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

5.2.47 instantiate

Instantiates a given composite model.

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

5.2.48 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)
```

5.2.49 listUnconnectedConnectors

Lists all unconnected connectors of a given system.

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

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

5.2.50 loadSnapshot

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

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

5.2.51 newModel

Creates a new and yet empty composite model.

```
status = oms.newModel(cref)
```
### 5.2.52 newResources

Adds a new empty resources to the SSP. The resource file is a “.ssv” file where the parameter values set by the users using “oms_setReal()”, “oms_setInteger()” and “oms_setReal()” are written to the file. Currently only “.ssv” files can be created.

The filename of the resource file is provided by the users using colon suffix at the end of cref. (e.g) “:root.ssv”

```python
status = oms.newResources(cref)
## Example
from OMSimulator import OMSimulator
oms = OMSimulator()
oms.newModel("newResources")
oms.addSystem("newResources.root", oms_system_wc)
oms.addConnector("newResources.root.Input1", oms.input, oms_signal_type_real)
oms.addConnector("newResources.root.Input2", oms.input, oms_signal_type_real)
## add Top level resources, the filename is provided using the colon suffix ":root.ssv"
oms.newResources("newResources.root:root.ssv")
oms.setReal("newResources.root.Input1", 10)
## export the ssp with new resources
oms.export("newResources", "newResources.ssp")
```

### 5.2.53 referenceResources

Switches the references of “.ssv” and “.ssm” in a SSP file. Referencing of “.ssv” and “.ssm” files are currently supported. The API can be used in two ways.

1. Referencing only the “.ssv” file.
2. Referencing both the “.ssv” along with the “.ssm” file.

This API should be used in combination with “oms_deleteResources”. To switch with a new reference, the old reference must be deleted first using “oms_deleteResources” and then reference with new resources.

When deleting only the references of a “.ssv” file, if a parameter mapping file “.ssm” is binded to a “.ssv” file, then the reference of “.ssm” file will also be deleted. It is not possible to delete the references of “.ssm” seperately as the ssv file is binded to a ssv file. Hence it is not possible to switch the reference of “.ssm” file alone. So inorder to switch the reference of “.ssm” file, the users need to bind the reference of “.ssm” file along with the “.ssv”.

The filename of the reference or resource file is provided by the users using colon suffix at the end of cref (e.g) “:root.ssv”, and the “.ssm” file is optional and is provided by the user as the second argument to the API.

```python
status = oms.referenceResources(cref, ssmFile)
## Example
```

(continues on next page)
from OMSimulator import OMSimulator
oms = OMSimulator()
oms.importFile("referenceResources1.ssp")
# delete only the references in ".ssd" file
oms.deleteResources("referenceResources1.root:root.ssv")
# usage-1 switch with new references, only ssv file
oms.referenceResources("referenceResources1.root:Config1.ssv")
# usage-2 switch with new references, both ssv and ssm file
oms.referenceResources("referenceResources1.root:Config1.ssv", "Config1.ssm ...")

5.2.54 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.

5.2.55 rename

Renames a model, system, or component.

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

5.2.56 reset

Reset the composite model after a simulation run.

The FMUs go into the same state as after instantiation.

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

5.2.57 setBoolean

Sets the value of a given boolean signal.

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

5.2.58 setCommandLineOption

Sets special flags.

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

Available flags:
## Usage

Usage: OMSimulator [Options] [Lua script] [FMU] [SSP file]

### Options:

- `--addParametersToCSV=<arg>` Export parameters to .csv file

- `--algLoopSolver=<arg>` Specifies the alg. loop solver

- `--clearAllOptions` Reset all flags to default

- `--deleteTempFiles=<bool>` Deletes temp files as soon as they are no longer needed

- `--directionalDerivatives=<bool>` Specifies whether directional derivatives should be used to calculate the Jacobian for alg. loops or if a numerical approximation should be used instead

- `--dumpAlgLoops=<bool>` Dump information for alg loops

- `--emitEvents=<bool>` Specifies whether events should be emitted or not

- `--fetchAllVars=<arg>` Workaround for certain FMUs that do not update all internal dependencies automatically

- `--help [-h]` Displays the help text

- `--ignoreInitialUnknowns=<bool>` Ignore the initial unknowns from the modelDescription.xml

- `--inputExtrapolation=<bool>` Enables input extrapolation using derivative information

- `--intervals=<int>` Specifies the number of communication points

- `--logFile=<arg>` Specifies the logfile (stdout is used if no log file is specified)

- `--logLevel=<int>` Specifies the log level (0 default, 1 debug, 2 debug+trace)

- `--maxEventIteration=<int>` Specifies the max. number of iterations for handling a single event

- `--maxAlgLoops=<bool>` Dump information for alg loops

- `--maxLoopIteration=<int>` Specifies the max. number of iterations for solving algebraic loops between system-level components

- `--mode=<arg>` Forces a certain FMI mode iff the FMU provides cs and me (cs, me)

- `--numProcs=<int>` Specifies the max. number of processors to use (0=auto, 1=default)

- `--progressBar=<bool>` Shows a progress bar for the simulation progress in the terminal

- `--realTime=<bool>` Experimental feature for (soft) real-time co-simulation

- `--resultFile=<arg>` Specifies the name of the output result file

- `--skipCSVHeader=<arg>` Skip exporting the scv delimiter in the header

- `--solver=<arg>` Specifies the integration method (euler, cvode)

- `--solverStats=<bool>` Adds solver stats to the result file, e.g. step size, not supported for all solvers

- `--startTime=<double>` Specifies the start time

- `--stepSize=<arg>` Specifies the step size

- `--stopTime=<double>` Specifies the stop time

- `--stripRoot=<bool>` Removes the root system prefix from all exported signals

- `--algLoopSolver=<arg>` Specifies the alg. loop solver for algebraic loops spanning over multiple components.

(continues on next page)
--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
--zeroNominal=<bool> Using this flag, FMUs with invalid nominal values will be accepted and the invalid nominal values will be replaced with 1.0

5.2.59 setFixedStepSize

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

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

5.2.60 setInteger

Sets the value of a given integer signal.

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

5.2.61 setLogFile

Redirects logging output to file or std streams. The warning/error counters are reset.
filename="" to redirect to std streams and proper filename to redirect to file.

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

5.2.62 setLoggingInterval

Set the logging interval of the simulation.

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

5.2.63 setLoggingLevel

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

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

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

5.2.65 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.

5.2.66 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)
```

5.2.67 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.

5.2.68 setSolver

Sets the solver method for the given system.

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

### 5.2.69 setStartTime

Set the start time of the simulation.

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

### 5.2.70 setStopTime

Set the stop time of the simulation.

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

### 5.2.71 setString

Sets the value of a given string signal.

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

### 5.2.72 setTempDirectory

Set new temp directory.

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

### 5.2.73 setTolerance

Sets the tolerance for a given model or system.

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

Default values are $1e^{-4}$ for both relative and absolute tolerances.

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

```python
oms_setTolerance("model", absoluteTolerance, relativeTolerance);
oms_setTolerance("model.root", absoluteTolerance, relativeTolerance);
```
Component, e.g. FMUs, pick up the tolerances from there system. That means it is not possible to define different tolerances for FMUs in the same system right now.

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

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

### 5.2.74 setVariableStepSize

Sets the step size parameters for methods with stepsize control.

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

### 5.2.75 setWorkingDirectory

Set a new working directory.

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

### 5.2.76 simulate

Simulates a composite model.

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

### 5.2.77 stepUntil

Simulates a composite model until a given time value.

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

### 5.2.78 terminate

Terminates a given composite model.

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

### 5.3 Example: Pi

This example uses a simple Modelica model and FMI-based batch simulation to approximate the value of pi.

A Modelica model is used to calculate two uniform distributed pseudo-random numbers between 0 and 1 based on a seed value and evaluates if the resulting coordinate is inside the unit circle or not.
The model is then exported using the FMI interface and the generated FMU can then be used to run a million simulations in just a few seconds.

Listing 1: Batch simulation of the simple Circle model with different seed values. All OMSimulator-related commands are highlighted for convenience.

```
import math
import matplotlib.pyplot as plt
import OMSimulator as oms

# redirect logging to file and limit the file size to 65MB
oms.setLogFile('pi.log', 65)

model = oms.newModel('pi')
root = model.addSystem('root', oms.Types.System.SC)
root.addSubModel('circle', 'Circle.fmu')

model.resultFile = ''  # no result file
model.instantiate()

results = list()
inside = 0

MIN = 100
MAX = 1000000
for i in range(0, MAX+1):
    if i > 0:
        model.reset()
    model.setInteger('root.circle.globalSeed', i)
    model.initialize()
    if model.getBoolean("root.circle.inside"):
        inside = inside + 1
    if i >= MIN:
```

(continues on next page)
The following figure shows the approximation of pi in relation to the number of samples.

![Approximation of pi](pi.png)

Fig. 1: Results of the above batch simulation which approximates the value of pi
This is a shared library that provides a OpenModelica Scripting interface for the OMSimulatorLib library.

### 6.1 Examples

```javascript
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 addBus

Adds a bus to a given component.
OMSimulator Documentation, Release v2.1.1.post155-gc803efc

6.2.2 addConnection

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

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

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

6.2.3 addConnector

Adds a connector to a given component.

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

The second argument “causality”, should be any of the following,

- "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 $\text{type}$, 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"

6.2.4 addConnectorToBus

Adds a connector to a bus.

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

6.2.5 addConnectorToTLMBus

Adds a connector to a TLM bus.

\[
\text{status} := \text{oms_addConnectorToTLMBus}(\text{busCref}, \text{connectorCref}, \text{type});
\]

6.2.6 addExternalModel

Adds an external model to a TLM system.
status := oms_addExternalModel(cref, path, startscript);

6.2.7 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.

6.2.8 addSubModel

Adds a component to a system.

status := oms_addSubModel(cref, fmuPath);

6.2.9 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"

6.2.10 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"
"OpenModelica.Scripting.oms_tlm_domain.oms_tlm_domain_hydraulic"
"OpenModelica.Scripting.oms_tlm_domain.oms_tlm_domain_electric"

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

"OpenModelica.Scripting.oms_tlm_interpolation.oms_tlm_no_interpolation"
"OpenModelica.Scripting.oms_tlm_interpolation.oms_tlm_coarse_grained"
"OpenModelica.Scripting.oms_tlm_interpolation.oms_tlm_fine_grained"
6.2.11 addTLMConnection

Connects two TLM connectors.

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

6.2.12 compareSimulationResults

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

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

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>

6.2.13 copySystem

Copies a system.

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

6.2.14 delete

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

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

6.2.15 deleteConnection

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

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

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

Deletes a connector from a given bus.

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

6.2.17 deleteConnectorFromTLMBus

Deletes a connector from a given TLM bus.

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

6.2.18 export

Exports a composite model to a SPP file.

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

6.2.19 exportDependencyGraphs

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

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

6.2.20 exportSnapshot

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

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

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

6.2.21 extractFMIKind

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

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

6.2.22 faultInjection

Defines a new fault injection block.

6.2. OpenModelica Scripting Commands
status := oms_faultInjection(cref, type, value);
The second argument type, can be any of the following described below

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

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

6.2.23 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

6.2.24 getBoolean

Get boolean value of given signal.

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

6.2.25 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);

6.2.26 getInteger

Get integer value of given signal.

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

6.2.27 getModelState

Gets the model state of the given model cref.

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

Get real value.

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

6.2.29 getSolver

Gets the selected solver method of the given system.

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

6.2.30 getStartTime

Get the start time from the model.

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

6.2.31 getStopTime

Get the stop time from the model.

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

6.2.32 getSubModelPath

Returns the path of a given component.

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

6.2.33 getSystemType

Gets the type of the given system.

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

6.2.34 getTime

Get the current simulation time from the model.

\[(\text{time, status}) := \text{oms_getTime(cref)};\]
6.2.35 getTolerance

Gets the tolerance of a given system or component.

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

6.2.36 getVariableStepSize

Gets the step size parameters.

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

6.2.37 getVersion

Returns the library’s version string.

```
version := oms_getVersion();
```

6.2.38 importFile

Imports a composite model from a SSP file.

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

6.2.39 importSnapshot

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

```
status := oms_importSnapshot(cref, snapshot);
```

6.2.40 initialize

Initializes a composite model.

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

6.2.41 instantiate

Instantiates a given composite model.

```
status := oms_instantiate(cref);
```
6.2.42 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.

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

6.2.43 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.

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

6.2.44 loadSnapshot

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

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

6.2.45 newModel

Creates a new and yet empty composite model.

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

6.2.46 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.

6.2.47 rename

Renames a model, system, or component.

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

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

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

6.2.49 setBoolean

Sets the value of a given boolean signal.

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

6.2.50 setCommandLineOption

Sets special flags.

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

Available flags:

| info: Usage: OMSimulator [Options] [Lua script] [FMU] [SSP file] |
|-------------------------|-----------------------------------------------|
| Options:                |                                               |
| - --addParametersToCSV=<arg> | Export parameters to .csv file         |
| - (true, [false])       |                                               |
| - --algLoopSolver=<arg> | Specifies the alg. loop solver              |
| - (fixedpoint, [kinsol]) | used for algebraic loops spanning over      |
| - multiple components.  |                                               |
| - --clearAllOptions     | Reset all flags to default                  |
| - --deleteTempFiles=<bool> | Deletes temp files as soon as             |
| - (true, [false])       |                                               |
| - --directionalDerivatives=<bool> | Specifies whether directional          |
| - derivatives should be used to calculate the Jacobian for alg. loops or |
| - if a numerical approximation should be used instead ((true), [false]) |
| - --dumpAlgLoops=<bool> | Dump information for alg loops             |
| - (true, [false])       |                                               |
| - --emitEvents=<bool>   | Specifies whether events should be emitted or not ((true), [false]) |
| - --fetchAllVars=<arg>  | Workaround for certain FMUs                |
| - (true), [false]       |                                               |
| - --help [-h]           | Displays the help text                      |
| - -ignoreInitialUnknowns=<bool> | Ignore the initial unknowns, |
| - from the modelDescription.xml (true, [false]) |                                               |
| - --inputExtrapolation=<bool> | Enables input extrapolation, |
| - (true, [false])       |                                               |
| - --intervals=<int> [-i] | Specifies the number of communication points (arg > 1) |
| - -logFile=<arg> [-l]   | Specifies the logfile (stdout is used if no log file is specified) |
| - --logLevel=<int>      | 0 default, 1 debug, 2 debug+trace          |
| - --maxEventIteration=<int> | Specifies the max. number of iterations for handling a single event |
| - --iterations [-i]     |                                               |
| - --communication points (arg > 1) |                                               |
| - --logFile=<arg> [-l]   |                                               |
| - --logLevel=<int>      | 0 default, 1 debug, 2 debug+trace          |
| - --debug+trace         |                                               |
| - --maxEventIteration=<int> | Specifies the max. number of iterations for handling a single event |
| - --iterations [-i]     |                                               |
--maxLoopIteration=<int> Specifies the max. number of iterations for solving algebraic loops between system-level components. Internal algebraic loops of components are not affected.

--mode=<arg> [-m] Forces a certain FMI mode iff 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

--skipCSVHeader=<arg> Skip exporting the scv delimiter in the header (true, [false])

--solver=<arg> Specifies the integration method (euler, [cvode])

--solverStats=<bool> Adds solver stats to the result file, e.g. step size; not supported for all solvers (true, [false])

--startTime=<double> [-s] Specifies the start time

--stepSize=<arg> Specifies the step size (<step size> or <init step, min step, max step>)

--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

--zeroNominal=<bool> Using this flag, FMUs with invalid nominal values will be accepted and the invalid nominal values will be replaced with 1.0

### 6.2.51 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);
```

### 6.2.52 setInteger

Sets the value of a given integer signal.

```
status := oms_setInteger(cref, value);
```
6.2.53 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
status := oms_setLogFile(filename);
```

6.2.54 setLoggingInterval

Set the logging interval of the simulation.

```c
status := oms_setLoggingInterval(cref, loggingInterval);
```

6.2.55 setLoggingLevel

Enables/Disables debug logging (logDebug and logTrace).

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

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

6.2.56 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.

6.2.57 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);
```
6.2.58 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.

6.2.59 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_mav2"
"OpenModelica.Scripting.oms_solver.oms_solver_wc_max"

6.2.60 setStartTime

Set the start time of the simulation.

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

6.2.61 setStopTime

Set the stop time of the simulation.

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

6.2.62 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);
```
6.2.63 setTLMSocketData

Sets data for TLM socket communication.

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

6.2.64 setTempDirectory

Set new temp directory.

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

6.2.65 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.

6.2.66 setVariableStepSize

Sets the step size parameters for methods with stepsize control.

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

6.2.67 setWorkingDirectory

Set a new working directory.

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

Simulates a composite model.

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

6.2.69 stepUntil

Simulates a composite model until a given time value.

```plaintext
status := oms_stepUntil(cref, stopTime);
```

6.2.70 terminate

Terminates a given composite model.

```plaintext
status := oms_terminate(cref);
```
OMSimulator has an optional dependency to OpenModelica in order to utilize the graphical modelling editor OMEdit. This feature requires to install the full OpenModelica tool suite, which includes OMSimulator. The independent stand-alone version doesn’t provide any graphical modelling editor.

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

See also FMI documentation and SSP documentation.

![OMEdit MainWindow and Browsers.](image)

**7.1 New SSP Model**

A new and empty SSP model can be created from *File-*->*New-*->*SSP* menu item.

That will open a dialog to enter the names of the model and the root system and to choose the root systems type.

**There are three types available:**

- TLM - Transmission Line Modeling System
7.2 Add System

When a new model is created a root system is always generated. If you need to have another system in your root system you can add it with `SSP->Add System`.

For example only a weakly coupled system (Co-Simulation) can integrate strongly coupled system (Model Exchange). Therefore, the weakly coupled system must be selected from the Libraries Browser and the respective menu item can be selected:

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

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:

The file browser will open to select an FMU (.fmu) or result file (.csv) as a subsmodel. Then a dialog opens to choose the name of the new sub-model.
7.3. Add SubModel

Fig. 4: OMEdit: Newly created empty root system of SSP model

Fig. 5: OMEdit: Add System

Fig. 6: OMEdit: Add System Dialog
Fig. 7: OMEdit: Add SubModel

Fig. 8: OMEdit: Add SubModel Dialog

Fig. 9: OMEdit: Root system with added FMU.
7.4 Simulate

Select the simulate button (symbol with green arrow) or select Simulation->Simulate from the menu in OMEdit to simulate the SSP model.

7.5 Dual Mass Oscillator Example

The dual mass oscillator example from our testsuite is a simple example one can recreate using components from the Modelica Standard Library. After splitting the model into two models and exporting each as an Model-Exchange and Co-Simulation FMU.

![Dual mass oscillator Modelica model (diagramm view) and FMUs](image)

Fig. 10: Dual mass oscillator Modelica model (diagramm view) and FMUs
Fig. 11: OMEdit: Simulate Dual Mass Oscillator SSP model
CHAPTER EIGHT

SSP SUPPORT

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

8.1 Bus Connections

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

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

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

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

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

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

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

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

**1D signal**

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

**1D physical (no interpolation)**

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

**1D physical (coarse-grained interpolation)**

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

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

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

3D physical (no interpolation)
### tlmType | causality
--- | ---
"state1" | output
"state2" | output
"state3" | output
"A11" | output
"A12" | output
"A13" | output
"A21" | output
"A22" | output
"A23" | output
"A31" | output
"A32" | output
"A33" | output
"flow1" | output
"flow2" | output
"flow3" | output
"flow4" | output
"flow5" | output
"flow6" | output
"effort1" | input
"effort2" | input
"effort3" | input
"effort4" | input
"effort5" | input
"effort6" | input

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

<table>
<thead>
<tr>
<th>timType</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>

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, 77
OMSimulator, 1
   Examples, 4
   Flags, 3
OMSimulatorLib, 4
   C-API, 5
OMSimulatorLua, 23
   Examples, 25
   Scripting Commands, 25
OMSimulatorPython, 42
   Examples, 43
   Scripting Commands, 44
OpenModelicaScripting, 62
   Examples, 63
   Scripting Commands, 63

S
SSP, 83
   Bus connections, 85
   TLM connections, 87
   TLM Systems, 86