MICM Quick Start

MICM is a package which is used within a CCPP-compliant host model. For this initial release, a simplistic box model (MusicBox) is being provided as the mechanism to drive MICM. For information on running MusicBox, please see the **MusicBox wiki page.**

A basic overview of the CCPP(V2) framework will be provided in this section. For more complete details, we direct the user to the **CCPP web site**.

At the heart of the plug-and-play capabilities of MICM within the CCPP framework is a scheme's metadata. This metadata is a table at the top of each scheme's three routines (init, run and finalize). The metadata table describes in precise detail each element for the routine's calling list. One element in this table is the standard_name and is what links each scheme's variable with variables in other schemes using the same standard name. We will walk-through an example table from the mozart chemistry driver scheme's run method. It is important to note that the format of the metadata will be changing with the next version of the framework, but the purpose of the metadata describing the interface variables will remain.

```
!> \section arg_table_chemistry_driver_moz_run Argument Table

!! | local_name | standard_name                                     | long_name                                |
units   | rank | type         | kind      | intent | optional |

!! |------------|---------------------------------------------------|------------------------------------------
|---------|------|--------------|-----------|--------|----------|

!! | vmr        | concentration                                     | species concentration                    |
mole/mole |    1 |real        | kind_phys | inout  | F        |

!! | TimeStart  | chem_step_start_time                              | Chem step start time                     |
s       |    0 | real         | kind_phys | in     | F        |

!! | TimeEnd    | chem_step_end_time                                | Chem step end time                       |
s       |    0 | real         | kind_phys | in     | F        |

!! | j_rateConst| photo_rate_constants                              | photochemical rates constants            |
s-1     |    1 | real         | kind_phys | in     | F        |

!! | k_rateConst| gasphase_rate_constants                           | k rate constants                         |
s-1     |    1 | real         | kind_phys | in     | F        |

!! | errmsg     | ccpp_error_message                                | CCPP error message                       |
none    |    0 | character    | len=512   | out    | F        |

!! | errflg     | ccpp_error_flag                                   | CCPP error flag                          |
flag    |    0 | integer      |           | out    | F        |

!!
```

The table is named with the method's name, in this case chemistry_driver_moz_run. The calling list for this method is (vmr, TimeStart, TimeEnd, j_rateConst, k_rateConst, errmsg, errflg). Each of these variables is completely described in the metadata. As described previously, the standard_name is the mechanism that links variables between several schemes. While MICM has been consistent with its standard names for this release, it is expected that they may change to become more precise as the framework's standard name library is developed. It is also important that the other fields be filled in correctly as the framework's interface provides consistency checking, and in the future may provide additional features such as unit conversion, array reordering, etc. based on the details provided.

It is important to note that the host model does not know the details about the calling list for chemistry_driver_moz_run. The host model knows about its own variables which have the same standard name, but it does not need to worry about the exact variables which are being passed nor the order in which they occur in the calling list. This allows another user to provide their own chemistry_driver run method and not need to change their calling list in any way. They simply document their own driver's calling list variables in a metadata table and their method is ready to be used within any framework-compliant host model.

Another section about the framework which needs to be described is the suite definition file. This file is the specification of which schemes are to be called and the order in which they are called. While the suite definition file resides in the host model, we will describe its use here as it is closely linked with MICM.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<suite name="MICM_terminator" lib="micmchem" ver="1.0.0">
  <group name="time_vary">
    <subcycle loop="1">
      <scheme>mass_quantities_util</scheme>
      <scheme>k_rateConst</scheme>
      <scheme>tuv_photolysis</scheme>
      <scheme>photolysis_interstitial</scheme>
      <scheme>chemistry_driver_moz</scheme>
    </subcycle>
  </group>
  <finalize>mass_quantities_util</finalize>
  <finalize>k_rateConst</finalize>
  <finalize>tuv_photolysis</finalize>
  <finalize>photolysis_interstitial</finalize>
  <finalize>chemistry_driver_moz</finalize>
</suite>
```

Here we take a look at the suite_MusicBox_terminator.xml xml file. Currently the framework had the init and run methods using the first group and the finalize method uses the last section of this xml file. The suite definition file is used to set up the calls to the named schemes in the order in which they appear. To replace the chemistry driver with the rosenbrock method (chemistry_driver_ros) it is as simple as substituting it's name in in place of chemistry_driver_moz.