

HarmoniIT

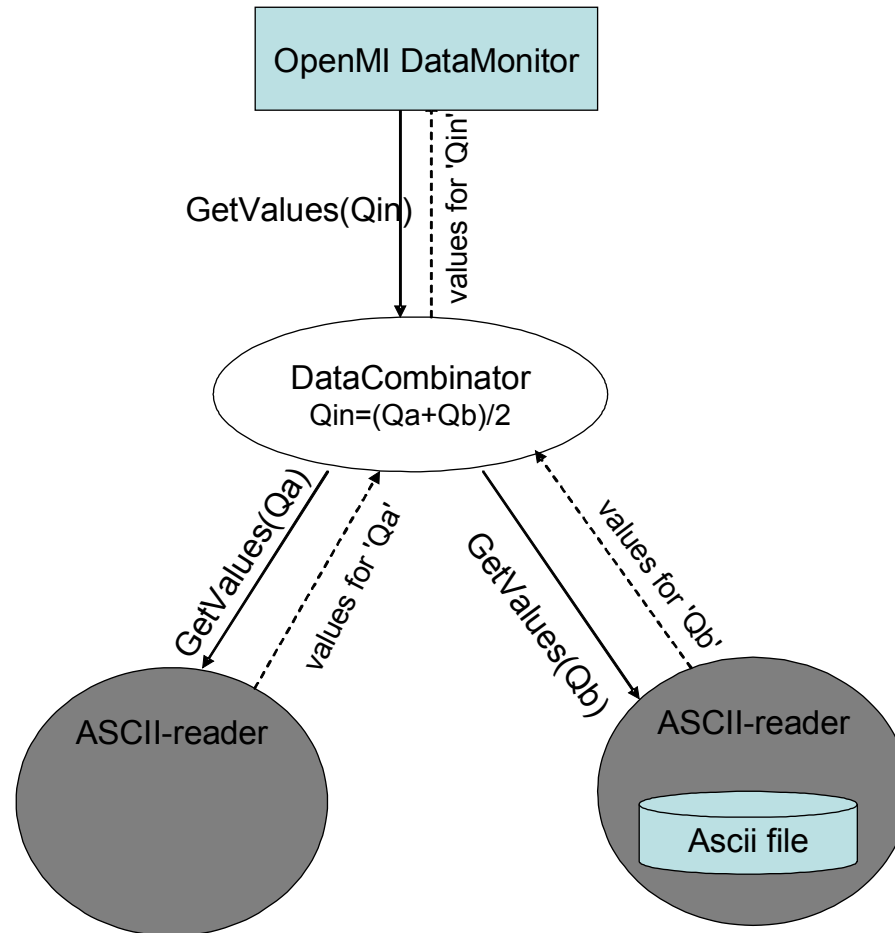


HarmoniIT

OpenMI Exercises

Rob Brinkman
WL | Delft Hydraulics

Exercises



Exercise 1:

A very simple linkable component



Step 1

Create a new solution and project in .Net

Step 2

Create a new class for your linkable component, called DataCombinator

Step 3

Inherit this class from the backbone linkable component

Step 4

Implement the GetValues method. Make a simple algorithm for filling the IValueSet using the incoming time

Step 5

Open a new project and write a little program to test your linkable component

Exercise 2:



Links, element sets and quantities

Step 1

Open your solution and projects from exercise 1

Step 2

Implement the AddLink method in the DataCombinator (register the link in an internal administration)

Step 3

Adjust GetValues so that for each element a value is returned

Step 4

Adjust GetValues so that only for the quantities "Q" and "h" values are returned

Step 5

Create an ElementSet and Quantity in your test program and use them when you call AddLink of the linkable component

Step 6

Transform your test program to NUnit and test it

Exercise 3:



Access to other linkable components

Step 1

Create a new linkable component, called AsciiReader, in your project.

Step 2

Write a method that reads an ascii file (...) and stores data internally

Step 3

Make sure this method is called before the first call to GetValues

Step 4

Implement GetValues so that it returns values read from the file

Step 5

Adjust your test program so that it creates a link between the DataCombinator and the AsciiReader

Step 6

Adjust the DataCombinator so that it uses data from the AsciiReader

Step 7

Test the new configuration using NUnit

Exercise 4: Configuring a linkable component



Step 1

Open the AsciiReader

Step 2

Implement the Initialize method. Demand an argument is passed containing the full path to the file to be read

Step 3

Adjust the DataCombinator so that it writes all calculated data to a file passed by an argument

Step 4

Make sure that the output file is open during the computation and that it will be closed afterwards

Step 5

Adjust the test program to pass the argument with the file name

Step 6

Test with NUnit and check the output file

Exercise 5: Exchange Items

Step 1

Open the AsciiReader

Step 2

Create an appropriate exchange item for the AsciiReader, based on the contents of the file

Step 3

Create an input and output exchange item for the DataCombinator. Use an element set of three elements

Step 4

Create omi files for the AsciiReader and the DataCombinator

Step 5

Start the user interface, create a solution and import the components. Check that you see the exchange items

Step 6

Create a composition with the AsciiReader and the DataCombinator. Add the link and fill in the start, step and end time

Step 7

Run the composition and check the output file

Exercise 6: Linkable Engine

Step 1

Create a new linkable component called SimpleRiver and inherit from LinkableEngine

Step 2

Create a class SimpleRiverEngine, which implements IRunEngine.

Step 3

Implement all methods of IRunEngine and make a simple computation in PerformTimeStep

Step 4

Use SetValue to provide input for PerformTimeStep and use GetValue to retrieve the results

Step 5

Implement SetEngineAPIAccess in SimpleRiver

Step 6

Create a test program in NUnit and test your implementation

Exercise 7:

Access FORTRAN code

Step 1

Open SimpleRiver.dsw in the Fortran IDE

Step 2

Run SimpleRiver and make sure you understand the code. Have a look at the output file

Step 3

Convert the code to a dll with the following methods: PerformTimestep, GetValues, SetValues, Initialize and Finish

Step 4

Adjust SimpleRiverEngine in .Net so that it calls the dll

Step 5

Test your dll with the NUnit test of exercise 6. Make sure the output file is still produced

Exercise 8:

Get Exchange Items from FORTRAN



Step 1

Open SimpleRiverEngine and let it inherit from IEngine

Step 2

All needed information should be retrieved from the FORTRAN dll. Make extra methods in the dll and implement them

Step 3

Create an omi file for SimpleRiver

Step 4

Import SimpleRiver in the user interface. Make sure the exchange items are populated as you expect them

Step 5

Create a composition where you use AsciiReader as input for SimpleRiver. Run the composition and check the output file

Exercise 9: Unit Conversion

Step 1

Open the DataCombinator

Step 2

Adjust the code of GetValues so that it delivers data in the right unit

Step 3

Adjust the test program to ask for a quantity which has a conversion factor not equal to one

Step 4

Identify a value as missing value (e.g. -999) and adjust the input. Make sure this value is processed correctly.

Step 5

Test the program using NUnit

Exercise 10: Events

Step 1

Open the DataCombinator

Step 2

Send an event of type `EventType.DataChanged` when a new value has been calculated. Make sure the calculated value has been stored

Step 3

Make sure the DataMonitor is compiled and registered in the GAC

Step 4

Open the user interface and create a new composition. Add the DataMonitor, the DataCombinator and the AsciiReader and let the DataMonitor get its values from the DataCombinator

Step 5

Set the DataCombinator as the start up component

Step 6

Run the composition. Make sure that the DataMonitor displays appropriate data

Exercise 11: Data Operations

Step 1

Adjust the DataCombinator so that it demands two input links. Validation check should be performed in Validate

Step 2

Adjust the calculation so that both input links are used. Provide two calculations: Average and Sum

Step 3

Let the linkable component provide two corresponding data operations

Step 4

Check in the user interface whether the data operations are interpreted as you expect

Step 5

Add an extra data operation Weighted Average, which uses an extra parameter a to compute result: $a * A + (1-a) * B$. Via the arguments the value of a is passed

Step 6

Check in the user interface whether results are as expected

Exercise 12: The Buffer

Step 1

Open the DataCombinator

Step 2

Add the buffer to the DataCombinator. Each time a value has been calculated, save it in the buffer

Step 3

Adjust your code so that values are read from the buffer if possible and otherwise continue with the computation until the requested values can be produced

Step 4

Adjust the test program so that values are asked in random order in respect with time

Step 5

Test your program

Exercise 13: Spatial Mapping

Step 1

Open the DataCombinator and find its element set

Step 2

Adjust the elements in the element set so that they have coordinates (x,y). Enter some values

Step 3

Open SimpleRiver and make sure the element set also has elements with coordinates

Step 4

Apply spatial mapping using the ElementMapper in Utilities
Read [PartF-org.OpenMI.Utilities technical documentation.pdf](#)

Step 5

Run your composition and inspect the results

Exercise 14: State Management

Step 1

Create a composition where the DataCombinator accepts output of the SimpleRiver and v.v. !

Step 2

Make sure there will not be an eternal loop by introducing a busy state in each linkable component

Step 3

Set busy to true within the GetValues call. When from outside a GetValues call is made to the component and it is busy, it is not allowed to call other linkable components

Step 4

Let the linkable component return a best guess when it is busy

Step 5

Test the composition and make sure there is no eternal loop