How to upgrade from version 1.4 IEngine

In version 1.4 a common case for an engine to become OpenMI compliant was to implement the IEngine interface. For many typical time progressing engines this was more in the lines of the model than implementing the ILinkableCompenent directly.

In version 2.0 the recommended pendant is to extend the abstract class LinkableEngine or (not recommended) the LinkableGetSetEngine.

The the IEngine of 1.4 and the LinkableEngine of 2.0 are very alike. The IEngine extends the IRunEngine. The LinkableEngine extends from the LinkableComponent directly. The LinkableEngine+LinkableComponent of 2.0 has approximately the same methods as the IEngine and IRunEngine of 1.4. Code written to the 1.4 IEngine can to a large degree be reused in the 2.0 LinkableEngine. Class diagrams from both 1.4 and 2.0 can be seen in the bottom of this document.

SetValues/GetValues

One of the major differences from 1.4 til 2.0 is the way values are get from and set to an engine. In 1.4 it was the IEngine that had the getValue and setValue functionality, based on a quantityId and an ElementsetId.

In 2.0 it is the exchange items that holds set/get-value functionality. How each exchange item sets or gets its data is up the the exchange item. It could hold a pointer to an unmanaged double array and set/get the value directly from that location. It could also store the value, and let the engine update the values at every timestep.

Together with the `LinkableEngine` is a number of EngineExchangeItems that can be used:

- EngineIIn/OutputItem uses an IValueSetter/IValueGetter interface to get/set the values.
- EngineDIn/OutputItem uses a delegate to get/set the values.
- EngineEIn/OutputItem works together with an LinkableGetSetEngine and delegates the work back to the LinkableGetSetEngine, very alike the 1.4 IRunEngine.
- EngineIn/OutputItem are abstract. Extending these and implementing the GetValuesFromEngine and SetValuesToEngine are required.

We recommend you to use the LinkableEngine in favor of the LinkableGetSetEngine: Even though the GetSet version is closer to the 1.4 ways of doing things and therefor will imply less work when upgrading to version 2.0, there are examples of bottlenecks due to the centralized GetSet method in the component.

Examples of implementing the IValueSetter/IValueGetter interfaces can be found in http://openmi.svn.sourceforge.net/viewvc/openmi/trunk/Oatc/src/csharp/Sdk/ModelWrapper/IValueSetter.cs

Example of using the IValueSetter/Getter version, where the output item is to return the i'th index of the `flowvector`:

```
    EngineIOutputItem outputItem = new EngineIOutputItem("id", quantity, elementSet, linkableEngine);
outputItem.ValueGetter = new ValueToVectorGetSetter<double>(flowvector, i);
```

The same example using the delegate version of the output item would look like

```
    EngineDOutputItem outputItem = new EngineDOutputItem("id", quantity, elementSet, linkableEngine);
int delegateIndex = i;    // required if i is updated in a loop
outputItem.ValueGetter = delegate()
    {
        IList res = new List<double>(1) { flowvector[delegateIndex] };
        return new ValueSet(new List<IList>{res});
    };
```

The nitty-gritty details for exchange items:

It is up to the implementor to decide whether data is stored in the exchange item at the end of a timestep, or whether data is retrieved from the engine on request, based on the `StoreValuesInExchangeItem` of the input and output item and the `DefaultForStoringValuesInExchangeItem` in the LinkableEngine. Each of the `EngineIn/OutputItems`, being either the I, D og G version, supports both. Both options have their implications. Note that any EngineInput/OutputItem will inherits its default value from the `LinkableEngine`.

In case data is stored in the exchange item, the LinkableEngine handles the data during its timestep. For an output item, if it has at least one consumer, then the exchange item data is updated at the end of each time step. If it has no consumers, the item is not used, and needs not be updated. For an input item, if it has a provider, its value is retrieved from the provider and stored to the engine before starting a new time step. This is the recommended approach.

In case data is not stored in the exchange item, but being get/set directly from/to the engine, the update is immediate. Be carefull with input items that adds up their contribution, i.e. an inflow. Updating an input item twice should not imply that the volume is added to the model twice, such an input item should store the values and let the engine do the update before the timestep.

Initializing

The initialization part from 1.4 can be reused to a large extent. Most changes has to do with classes having changed their names and arguments.

Events

Whenever an exchange item changes state, the component should trigger an event containing a `ExchangeItemEventArgs`. This has to be done for every active exchange item. Often the exchange items changes at the end of the timestep, hence the event should be triggered there. However, some exchange items may only change their value when they are actually requested for data, in which case the event should be triggered at every request that changes the data.

Extending the LinkableEngine

When extending the LinkableEngine, the following abstract methods needs to be implemented

LinkableEngine

```
    // Time Info and Time stepping
protected internal abstract ITime StartTime { get; }
protected internal abstract ITime EndTime { get; }
public abstract ITime CurrentTime { get; }
public abstract ITime GetCurrentTime(bool asStamp);
public abstract ITime GetInputTime(bool asStamp);

// Model control
public abstract void Initialize(IArgument[] arguments);
protected internal abstract string[] OnValidate();
protected internal abstract void OnPrepare();
protected internal abstract void PerformTimestep(ICollection<EngineOutputItem> requiredOutputItems);
public abstract void Finish();
```

LinkableGetSetEngine

```
    public abstract IValueSet GetEngineValues(ExchangeItem exchangeItem);
public abstract void SetEngineValues(EngineInputItem inputItem, IValueSet values);
```

Generic step by step instructions when upgrading

This section will describe the steps involved when upgrading from 1.4 to 2.0. It is assumed that the source is a Microsoft Visual studio solution, containing one C# project with the OpenMI wrapping of the engine, we will here call it `MyEngineWrapper.csproj`. It is also assumed that the engine was wrapped according to the 1.4 guidelines, i.e., the `MyEngineWrapper.csproj` project contains files alike:

- `MyModelWrapper.cs` - implementing the IEngine interface
- `MyModelLinkableComponent.cs` - extending the LinkableEngine

And for unmanaged engines also the two files:

- `MyModelDLLAccess.cs` - accessing the unmanaged dll
- `MyModelDotNetAccess.cs` - wrapping the `MyModelDLLAccess` to follow C# calling convetions

Prepare solution

- Download the OpenMI 2.0 SDK source code and the standard
- Convert existing solution to a Microsoft Visual Studio 2008 solution.
- Remove all 1.4 projects from the solution
- Add the following projects to the solution
  - `OpenMI.Standard2/OpenMI.Standard2.csproj`
  - `Oatc.OpenMI/SDK/Oatc.OpenMI.Sdk.csproj` - this includes Backbone, Buffer and Spatial (only one project now)
  - `Oatc.OpenMI/ModelWrappers/EngineWrapper/Oatc.OpenMI.Sdk.ModelWrapper.csproj`

Prepare engine project - MyEngineWrapper.csproj

- Change the project to use at least version 3.0 and preferably 3.5 of the .Net framework.
- Remove reference to all 1.4 projects
- Add references:
  - `OpenMI.Standard2`
  - `Oatc.OpenMI.SDK`
- Remove the file `MyModelLinkableComponent.cs`, if existing. This file is no longer needed.

Generally no changes are required to the `MyModelDLLAccess.cs` and the `MyModelDotNetAccess.cs`, if existing.

Changes to the `MyModelWrapper.cs`

- Adjust the namespace lines, from `using OpenMI.Standard` to `using OpenMI.Standard2`
- Change the class definition from `public class MyModelWrapper : IEngine` to `MyModelWrapper : LinkableEngine`, or alternatively the `LinkableGetSetEngine`

We recommend you to use the `LinkableEngine`, since it is implemented following the ideas in version 2.0. You may use the `LinkableGetSetEngine`, which is more in the lines of the 1.4 version, and suffers from the problems related to the centralized Get/Set methods in the component to handle all data transfer. However, the `LinkableGetSetEngine` may give rice to less changes to the existing code, and can be used in the first iteration.

- If the implementation of the functions `GetInputExchangeItem` and `GetOutputExchangeItem` is creating exchange items, move the implementation to the `Initialize` method, now instead populating the lists of `EngineInputItems` and `EngineOutputItems`.
- The methods `GetInputExchangeItem`, `GetInputExchangeItemCount`, `GetOutputExchangeItem` and `GetOutputExchangeItemCount` can deleted.
- All exchange items must be added to the the lists `EngineInputItems` and `EngineOutputItems`. Any existing private field defining the lists can be deleted.
- All input and output exchange items must be one of the EngineInputItem or EngineOutputItem mentioned earlier.

- Implement the missing inherited abstract methods.
- Clean up in the methods that are no longer used/required, like the `GetTimeHorizon` and `GetEarliestNeededTime`

Changes in the SDK classes compared to 1.4

- `Time`: Only one class handling as well TimeStamp as TimeSpan (time interval). No longer need for the CalendarConverter, the Time class contains these tools now.
- `Unit`: The `ID` from 1.4 is now the Caption. There is a number of predefined units, and you are welcome to extend on the list.
- `Quantity`: The `Dimension` property is moved to the `Unit` of the quantity. The `ValueType` is now a build in type, for example `typeof (double)`.
- `ElementSet`: `Id` is now `Caption`. The `SpatialReference` is renamed to `SpatialReferenceSystemWkt` and is a string.
- `ElementType`: Has changed. Find the one that suites the best.

Class diagrams

The 1.4 IEngine

blocked URL

The 2.0 LinkableEngine

blocked URL