ECO - EcoModeler Quick start

Table of Contents

Getting started 1

Layout 2

Creating a class diagram 3

Adding a class 4

Adding a method 5

Adding a property 6

Inheritance 8

Adding class associations 9

Self referencing associations 11

Adding association classes 11

Adding a state machine diagram 14

States 14

Transitions 15

Composite states 16

Defining interfaces and realization 18

Tidying up diagrams 20

Advanced modeling techniques 22

External references 22

Migrating Together models 24

Generated source code 25

Deleting model elements 28

Setting project options 30

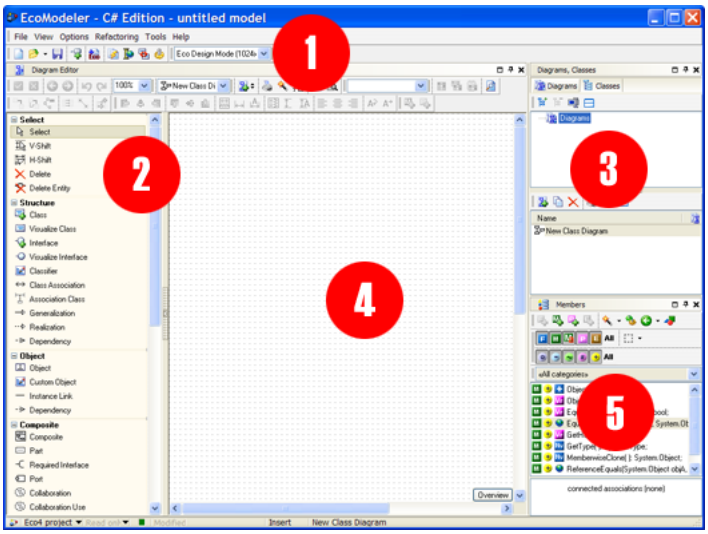
Index a

1 Getting started

To launch EcoModeler from VisualStudio right-click the model file (*.mmcseco) and select "Open in EcoModeler" from the context menu.

2 Layout

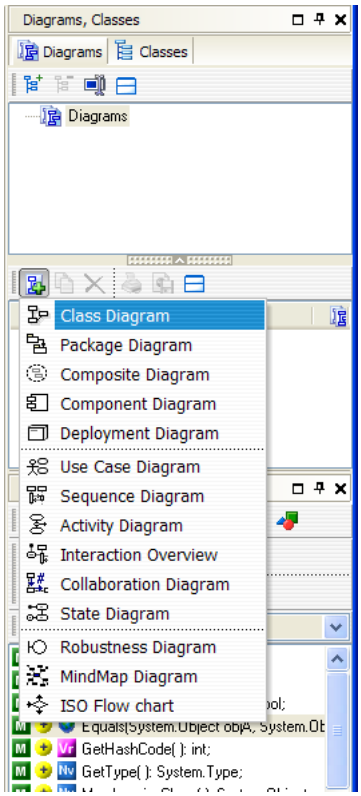
EcoModeler supports layout customization. Areas of the user interface may be dragged and docked to edges of the main window or within other areas to form tabbed controls, these custom layouts may then be saved under a specified name and then selected via a combo box at the top of the application. The default layout is as follows:



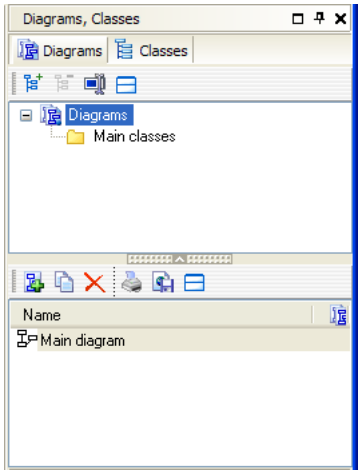
1. The combo box that selects which layout to use.
2. A selection of diagram types that may be created. The elements displayed here change when a diagram is selected.
3. A list of diagrams created within this project file (bottom half), these may be categorized into folders and sub folders (top half). Note that these are not folders on disk but merely a form of categorization within the project. The tab at the top of this area allows you to switch between a view of diagrams within the project and a hierarchical tree of all classes in the project.
4. The client area in which the selected diagram will appear.
5. A list of members belonging to the most recently selected class in the project.

3 Creating a class diagram

Diagrams are added to the project by clicking the *Add Diagram* button.



A dialog will appear asking for a name for the new class diagram. Once created the diagram will appear in the diagram list, this diagram may be drag/dropped into different categories if you have added any.



3.1 Adding a class

Select the newly created class diagram and the client area in the middle of the application will show the design surface for the diagram. Right-click the design surface and select the *Add* menu, and then the *Class* sub menu. A dialog will now appear showing the properties of the new class to create.

Class tab

- Class name: The name of the class.
- Ancestor: The class from which this class will descend. It is also possible to specify the ancestor class via the diagram design surface.
- Style: Specifies whether the class is Concrete or Abstract.
- Persistency: By default this will contain the value "Auto detect" which is equal to specifying that the class is persistent. You may optionally specify that a class is Transient here.

ECO tab

This tab contains the various ECO specific settings for the class including:

- Default string represention: The Object Constraint Language (OCL) expression to evaluate in order to produce a string that should be returned when the object is evaluated with the OCL expression AsString.
- Optimistic locking: The locking strategy to use for this class.
- Table mapping: Whether this class should have its own table in the database, or merge its columns into its parent/child class's table. This is only applicable when you allow ECO to automatically generate the database structure on your behalf.
- TableName: You may specify a specific table name to use when generating the database. The constant string <NAME> will be replaced with the name of the class, so T_<Name> on a class named "User" would result in a table named T_User.
- Database: When the ECO application works with more than one database to persist objects this value specifies which database alias should be used to persist instances of this class to.
- Former names: Whenever you rename this class you may specify the original name here. When you evolve the database structure the data from the original table will then be copied over to the new table.
- Constraints: Invokes an editor to specify OCL constraints for this class.
- Derivation expressions: Invokes an editor allowing you to specify OCL expressions which will override the expression to use for any OCL derived members defined on any of this class's superclasses.
- Tagged values: Invokes an editor allowing you to specify additional meta data for this class, these tagged values may be read by your application at runtime.

Symbol style tab

These settings apply to the diagram element and not the class itself. The same class may be added to the diagram multiple times and each element on the diagram for the same class could have different settings within this tab.

- Auto member compartments style:
  - Diagram defined style: Uses the same setting as on the diagram.
  - Auto maintained member list: Allows you to set which types of members to appear, properties, methods, exposure level, etc.

- Custom auto member list: Creates an addition tab "Custom members" in which you may specify exactly which members to display.
- Custom negated auto member list: Shows all members of the class *except* for the members specified in the Custom members tab.

- Auto size Width/Height: By default these are checked, as a result the element on the diagram representing the class will auto size to accommodate its members. Unchecking them will allow you to manually resize the element on the diagram. This can be useful when you wish to display many associations between two classes.

Documentation tab

This tab allows you to enter a one-liner and full description for the class. It is possible to emit this documentation by setting the documentation setting in the project options dialog.

Hyperlinks tab

This tab allows you to add hot links to your diagram element. These links can link you to any diagram in the project and appear as a small icon in the top of the element on the diagram.

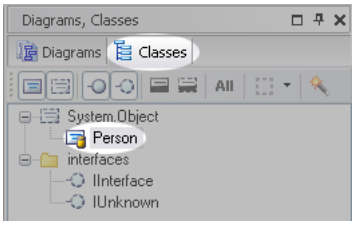
Attributes tab

This tab allows you to specify additional .NET attributes to attach to the class during code generation.

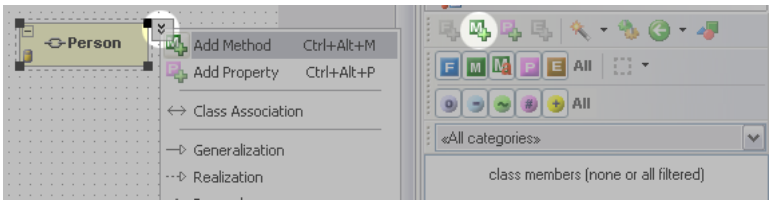
3.1.1 Adding a method

Methods may be added to your class, during code generation an empty method stub will be created. These methods may be executed as usual by invoking them from compiled code. In addition it is possible to invoke these methods from an expression by using the Action Object Constraint Language (Action OCL), and also as a result of state changes within a state machine attached to an ECO class.

To add a method first select the class by single-clicking it on the class diagram surface, or select the [Classes] tab and select it by single-clicking it in the tree view control.



Once selected you may now add a method by either clicking the M+ icon above the list of class Members, or by clicking the quick menu next to the class on the diagram surface and selecting Add Method.



Method tab

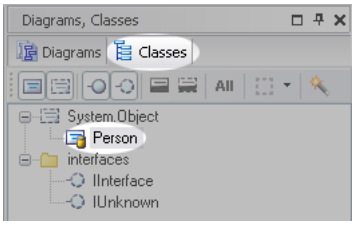
- Name: The name of the method.
- Method type: This should be set to "method".
- Parameters: This is a list of parameters the method signature should consist of. You may add/remove/modify parameters individually in the area directly beneath, or toggle the button directly beneath the label to edit the parameters as a single string.
- Return type name: Specifies the type of the object the method should return. Note that to return instances of modeled ECO classes you should use a derived (calculated) association instead of a method, otherwise the method will not be compatible with the OCL evaluator.
- Visibility: Specifies the encapsulation level to apply to the method signature.
- Binding kind: Specifies whether the method should be marked override, virtual, etc. Binding kind is nothing to do with .NET data binding.
- Inheritance restricted: If this method overrides a method in an ancestor class then checking this check box will bind the method signature to that of the ancestor's method, any changes made to the ancestor's method's signature will also change this method signature.

ECO tab

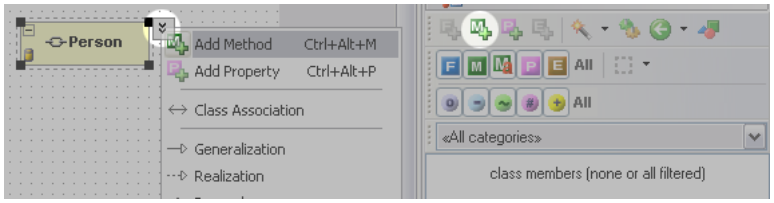
- Query method: If this check box is checked it indicates that this method does not affect the state of its parent object when executed, meaning that no property values are modified etc. This exposes the method to the OCL expression evaluator and is therefore accessible from more places within ECO (such as handles, derived member expressions, and so on). This is unlike non-query methods which are only available to the Action OCL evaluator.
- Trigger method: This check box indicates that when invoked this method will trigger a change of state in the object's state machine. If a method is a trigger then no custom source code may be implemented within the method's implementation.
- Non-ECO method: This check box indicates that the method should not be available to any of the OCL evaluators. This allows you, for example, to have overloaded methods with the same name, which is not permitted by the ECO OCL evaluator. You may wish to add such methods to the class's source code directly, this feature exists for those who wish to show the method for illustrative purposes.
- Pre-condition: This should be empty, or contain an OCL expression that returns a boolean. If present on a trigger this expression is evaluated before permitting the trigger to execute. The result of the expression may additionally be evaluated with the expresion "self.MethodName?" or "self.MethodName?(parameters)".
- Post-condition: This item is obsolete.
- Body: If not present then the method must be implemented in source code. If an expression is present it will be executed with the Action OCL evaluator whenever the method is invoked, in which case no custom source code may be implemented within the method.
- Tagged values: The Edit button invokes an editor in which you may attach tagged values to the method which may be read during runtime.

3.1.2 Adding a property

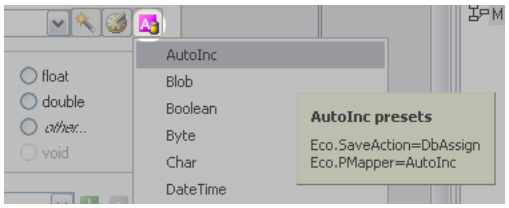
To add a property (UML Attribute) first select the class by single-clicking it on the class diagram surface, or select the [Classes] tab and select it by single-clicking it in the tree view control.



Once selected you may now add a property by either clicking the P+ icon above the list of class Members, or by clicking the quick menu next to the class on the diagram surface and selecting Add Property.



- Name: The name of the property, this must be unique.
- Visibility: Specifies the encapsulation level of the property when code is generated.
- Type name: The .NET type of the value this property will hold. You may type in the name directly or choose from the pick list directly beneath. In addition you may also select a type from the ECO Type Presets button to the very right of the control. Selecting a type from this list will not only set the property type but will in addition set various other settings, such as in the illustration where the SaveAction and PMapper values are set so that the property will act as an auto incrementing value assigned by the RDBMS.



- Attribute type:
  - Persistent: The value will be saved to and retrieved from the data storage when an instance of this class is persisted or retrieved.
  - Transient: The value will not be saved or retrieved with the object instance.
  - Derived (read-only): The value is calculated on demand via an OCL expression or a specially named method in the class.
  - Derived (read / write): In addition to being derived, it is also possible to set the calculated value. ECO will look for a specially named method which will be executed in order to set the value, this method is responsible for parsing the value and acting appropriately. For example, a derived FullName property might parse "Mr/Peter/Morris" and update Title, FirstName, and LastName.
  - Derivation OCL: If the property is marked as Derived then you may enter an OCL expression to be evaluated when determining the value of the property at runtime. You may invoke the OCL editor window by clicking the "OCL" button to the right of this text box. If no OCL expression is entered ECO will look for a method with the following signature at runtime in order to determine the value of the property.

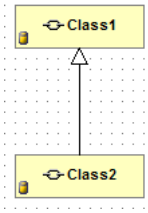
```
private object PropertyName_DeriveAndSubscribe(ISubscriber reevaluateSubscriber,
ISubscriber resubscribeSubscriber)
```

- Column name: If the property is persistent this will determine what column name to use in the database to store its value. <Name> will evaluate as the property name. For example, X_<Name> on a property named "FirstName" would result in a column named X_FirstName.
- Initial value: Specifies the initial value to be assigned to this property whenever a new instance of the class is created. This does not affect instances recreated as a result of retrieving it from the data storage.
- Default DB value: Specifies the "DEFAULT" value clause to use when creating the column in an RDBMS data storage.
- Length: When the property is a type that has variable lengths (such as strings) this value identifies the maximum length permissable. This value is used to specify column sizes when using an RDBMS data storage.
- Persistence mapper: If the property is of a custom type (such as System.Drawing.Font) then a custom persistence handler is required in order to:
  1. Create a column of the appropriate type in the class's table.
  2. Save the value to the column.
  3. Recreate the .NET value based on data retrieved by ECO from a DB column value.
- Save action:
  - None: The default behavior.
  - Freeze: The property is read/write when created, but becomes read only once the instance has been persisted.
  - DbAssign: The property is read only because it is assigned by the persistent storage. An example of this is when you select the ECO pre-defined data type AutoInc which creates an auto-incrementing column. Once the object has been persisted ECO will set the value of the property according to the value generated by the data storage.
- Tagged values: The Edit button invokes an editor allowing you to add named meta-data values to the property. These values may be read at runtime.
- Former names: The Edit button invokes an editor allowing you to specify a list of string values. If your model consists of a Person class that has amongst its members a property named "ChristianName", at some later point you may wish to rename this to a more generic "FirstName". When you then invoke the database "Evolve" feature on the ECO space to synchronize your database structure with your model you will find that the ChristianName column is dropped and a new column named "FirstName" is created without any values in any of the rows. In order to keep the data you would specify ChristianName as a Former Name so that when ECO evolves the database structure it ensures the old values in the ChristianName column are copied across to the new column before the old column is dropped.
- ECO Options:
  - Allow Null: If this ticked then the property may be assigned a null value, this includes via the Action OCL evaluator.
  - Generate as Nullable: If ticked then the property is created as a Nullable<T> property so that it may easily be assigned the value "null" in code. Allow Null must be ticked in order to enable this check box.
  - Delayed fetch: If ticked then the value of the property will not be fetched from the data storage when the object instance is retrieved. Any attempt to read the value of the property will first retrieve the value from the data storage. This feature is useful for large property values such as Images or other BLOB type columns where the property isn't always used by the application. For example, the Signature property on a DeliveryNote might only be displayed when the user clicks a "View Signature" button on a form. This feature saves network bandwidth and memory consumption.
  - State attribute: This check box indicates that the property is used to store the state of a state machine attached to the class, or a logical region within the state machine.
  - Has User Code: Ordinarily property values read via OCL expressions will bypass the object instance and go directly to the internal cache of the EcoSpace. If the property has specialized code in the getter or setter of the property then tick this check box to ensure that ECO directs all property operations through the property on the object rather than going straight its internal cache.

3.1.3 Inheritance

There are two ways of specifying the base class of a class within the model.

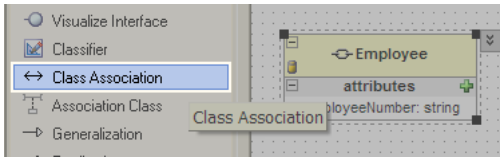
1. Double-click the class and specify a class in its Ancestor combo box.
2. Select the "Generalization" item in the tool palette. Then click the descendant class, hold the mouse button down, move the mouse cursor over the base class, and then release the mouse button.



3.2 Adding class associations

Class associations are a way of holding a property reference to an instance of another modeled ECO class. A an association may be one-way where a property is added only to one of the two classes involved, or two-way where a reciprocal property is added to the target class. To add an association between two classes:

1. First select a diagram on which you wish to visualize the association.
2. Select "Class Association" from the tool palette.



3. Click the class on the diagram and hold the mouse button down. As you move the mouse you will see a dotted line drawn from the source class to the mouse cursor. Keep the mouse button held down, move the mouse cursor over the target class and then release the mouse button.
4. A dialog window will now appear allowing you to enter details about the association before it is finally added to the model.

Association

- Name: The name of the association. This must be unique within the model and is not the name of either of the properties generated. I like to use the pattern [Child class name][Property name to parent class], for example PurchaseOrderLineOrder.
- Association class: This is explained later.
- Embedding: When the association is persistent (Association type is "Persistent" and the classes at both ends of the association are also persistent) ECO will need to know which table in an ECO generated database will contain a column referencing the other table.
  - None: Neither table will have a reference column added, instead a table with the same name as the association "PurchaseOrderLineOrder" will be created with a column for the ID of each end of the association; PurchaseOrder and PurchaseOrderLine.
  - End 1 - Store EmployeeID in Department table: The table created for the Department class will contain a column referencing the other table.

    - End 2 - Store DepartmentID in Employee table: The table created for the class at the other end of the association will contain the referencing column.
- Association type
    - Persistent: The association information is read from and written to the persistent storage.
    - Transient: The association information exists in memory only and does not exist beyond the life of the EcoSpace.
    - Derived: The object(s) at the end of the association are deduced by either evaluating an OCL expression or by executing a specially named method on the class.
- Former names: Whenever the name of the association is changed it is possible to record a list of former names. When an existing database structure is "evolved" to accommodate the new model structure an existing link table will be renamed.

Note:

- For multi---multi associations no embedding is possible, a link table will always be required.
- For single---multi associations the only valid choices are None or to embed the single end into the class on the multi end of the association.
- For single---single associations all combinations are valid.

End 1

- Source class / End point class: These read only items identify which class the current association end is attached to. For example, if the Source class = "Department" and the End point class = "Employee" then changes to the settings on this tab will be reflected on the property Department.Employees.
- Visibility: Specifies the encapsulation level to apply to the property's signature.
- Column name: Specifies the name of the column to generate in the persistent storage. <Name> will be replaced with the role name, so X<Name> would evaluate as XDepartment.
- Style: This specifies how this end of the association line should be drawn on the diagram. In addition to being a visual aid it also specifies how to act when deleting an object that has associated objects. See "Delete action".

Image	Description
	Undefined: Objects will be unlinked.
	Aggregation: Deletion will be prohibited if there are any object instances at the other end of the association.
	Composition: Any object instances at the other end of the association will also be deleted.
	Navigable: The association is navigable in this direction, a property will be added to the class on the other end of the association.
	Non navigable: The association is not navigable in this direction, no property will be added to the class on the other end of the association.

Note: Not specifying a navigability will imply that navigation is possible. The exception to this is when only one end is marked as navigable, this implies that the association may only be navigated in one direction.

- Default region mode: TODO
- Role: The name of the property to add to the class when generating source code. This may be left empty if this end of the association cannot be navigated to from the opposite end of the association.
- Delete action: Specifies how to act when an object at the other end of the association is deleted. When "Default" the behavior is inferred from the "Style" setting. Setting it to any other value will perform the following actions:
    - Allow: The objects will be unlinked.
    - Prohibit: The object at the other end of the association will not be allowed to be deleted.
    - Cascade: The object at the current end of the association will be deleted along with the parent object at the other end of the association.

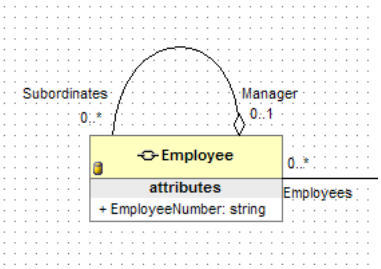
- Multiplicity: Specifies how many object instances may be referenced at this end of the association. Anything with an upper bound of "*" will result in a list property being added to the class at the opposite end of the association.
- Save action:
  - None: The default behavior.
  - Freeze: The property is read/write when created, but becomes read only once the instance has been persisted.
  - DbAssign: The property is read only because it is assigned by the persistent storage.
- Ordered: This option is only valid when the current end of the association has an upper bound that is greater than 1 (a list of objects / a multi role). When checked it will cause an additional column to be added to the persistence storage so that the order of the objects within the list may also be persisted. This is useful for recording the order in which items are stored in a list where there is no property on the class to order by. It is not intended to be used for example on a PurchaseOrderLine which does have an OrderLine property which may be used to visually order the collection in a GUI.
- Has user code: If this checkbox is checked ECO knows that you intend to write code in the property that should be executed whenever the property is accessed via an OCL evaluation.
- Derivation OCL: If the association is marked as Derived this input will allow an OCL expression to be entered which should be evaluated in order to determine the list of objects that should be included in the list when the value of the property is read. If the association is Derived and no OCL is entered ECO will look for a specially named method in order to determine the list of objects.
- Former names: As with the Association Name this allows you to record name changes so that values in the persistent storage may be kept when "evolving" the data structure.
- Tagged values: The Edit button invokes an editor allowing you to add named meta-data values to the property. These values may be read at runtime.

End 2

This is exactly the same as End 1, except the Source class is the class at the other end of the association.

3.2.1 Self referencing associations

A self referencing association is an association where both ends originate and end on the same class.

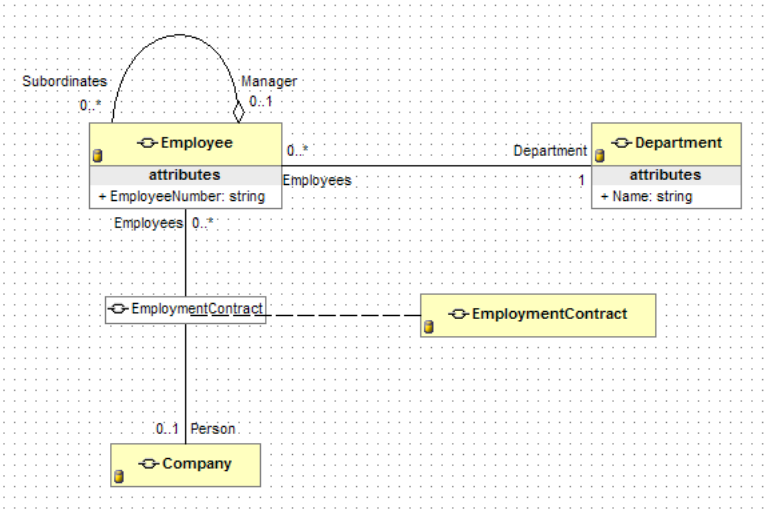


To add a self referencing association click the "Class Association" item on the tool palette, and then single-click the relevant class on the diagram without holding down the mouse button. This will add an association where both ends are attached to the same class.

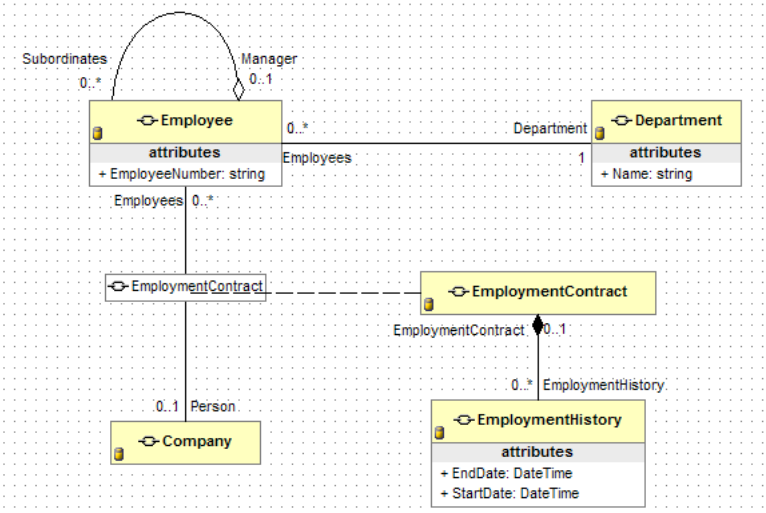
3.2.2 Adding association classes

An association class is a special kind of ECO class that holds additional information about associated classes without having to store that information in either of the classes in question. For example an association between Company (Employer : 1)

and Employee (Employees : 0..*) you may wish to add additional information in a class EmploymentContract.

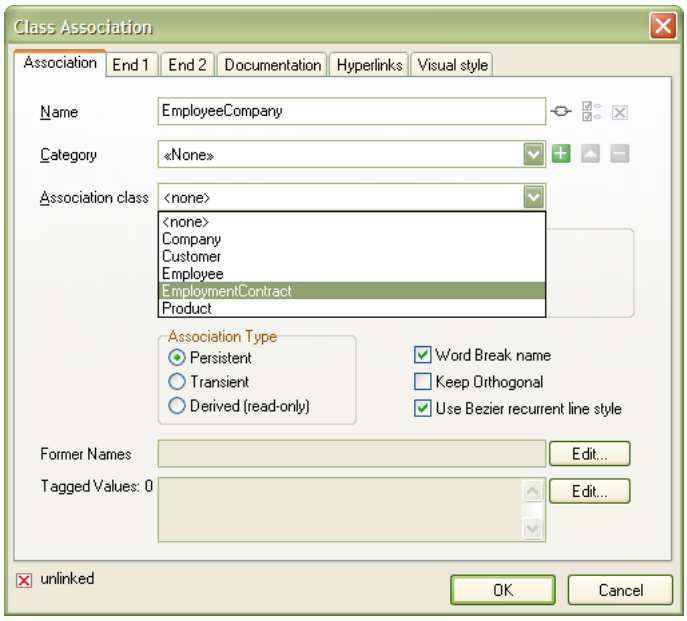


In this case any additional properties such as EmploymentStartDate / EmploymentEndDate could quite easily be added to the Employee class, however if the same person were to be employed more than once over the years you would need to create more than once instance of the Employee class for the same person. Instead you may wish to record employment history details via an association class.

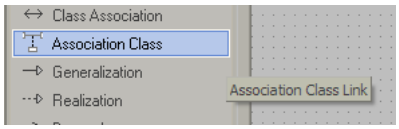


To specify an association class you have two options:

1. Double-click the association and then select a class in the "Association class" drop down list.



2. Select the Association Class item in the tool palette
   1. Click the class to become the association class and hold the mouse button down.
   2. Move the mouse cursor over the association line.
   3. Release the mouse button to complete the line.

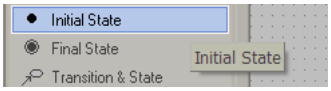


4 Adding a state machine diagram

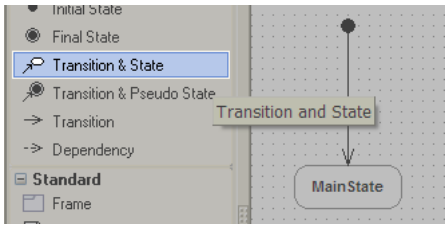
To add a state machine diagram right-click the class in question and select "Associated State Diagram" on the context menu and then select "Create State Diagram". This will create a state diagram that is associated with this class in the model. Selecting the "Associated State Diagram" again will enable "Edit State Diagram" instead.

Note: State diagrams do not create entities in the model. This means that, unlike classes, if you delete a state from a state diagram it is also deleted from the model.

1. Delete the elements on the diagram (initial state, transition, final state).
2. Add an initial state to the diagram from the tool palette.



3. Now select Transition & State from the tool palette. Click the mouse on the Initial state on the diagram, hold the mouse button down, then move the mouse cursor to an empty space on the diagram and release the mouse button.



4. You may change the name of the state by single clicking its name label and changing via the in-place editor.

State machine rules:

- There must be exactly one Initial state.
- The state machine diagram must have a transition that enables ECO to immediately leave the initial state.
- The state machine may have zero or many Final states. If a final state is entered the object instance is deleted.

4.1 States

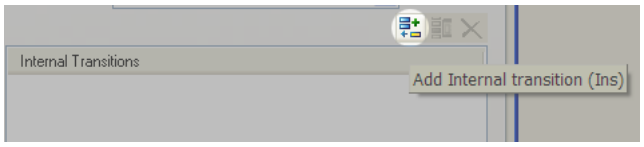
Double-click a state to bring up its property dialog.

State symbol

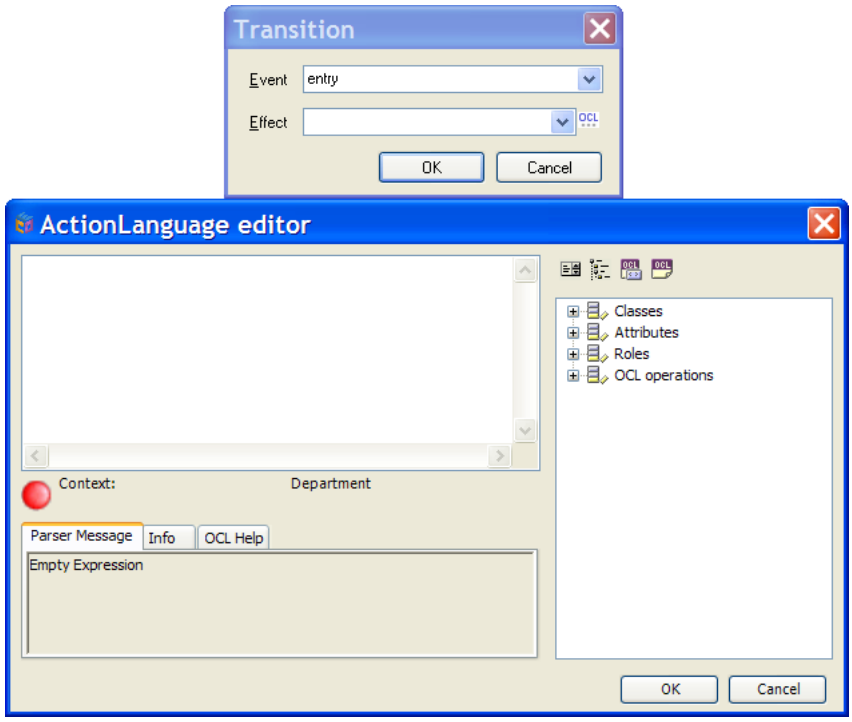
- Name: This is the name of the state. The format of the name is restricted in the same way a variable name would be. E.g. it must start with an alpha character, must not contain spaces, and so on.
- Representation: This free text value can be used to provide a more human readable value such as "In progress".

- State type: See composite states (see page 16).
- Internal transactions: This lists which actions (Entry / Exit) have been assigned.

To add an internal transition click the "Add internal transition" button above the "Internal Transitions" list, or press the INS key.



You may select the transition type using the "Event" combo box, and then enter the ActionLanguage expression in the "Effect" text box. You may invoke the ActionLanguage editor dialog by clicking the "OCL" button to the right of the text box.



You may enter multiple lines of action language script which will be executed when the state is entered or exited, depending on which Event you link the script to.

State rules:

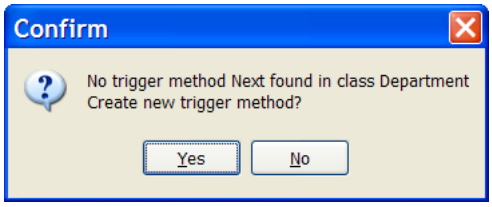
- The name of the state must be unique within its container. A container may either be the diagram itself or a region within another state (sub state machine).

4.2 Transitions

To add transitions select "Transition" from the tool palette. Then click the source state, hold down the mouse button, and drag the mouse cursor onto the destination state before releasing the mouse button. If you wish to add a new state and a

transition to the state you may wish to use "Transition & State" in the tool palette instead. Double-click the transition to bring up its property dialog.

- Trigger: If the transition is to be invoked from a method on the class the name of the method should be selected here. Only methods with "Trigger Method" checked will be displayed in the list. You may add a trigger method without any parameters by entering the name of a method that does not exist, when you click the "OK" button you will be asked if you wish to create a trigger method.



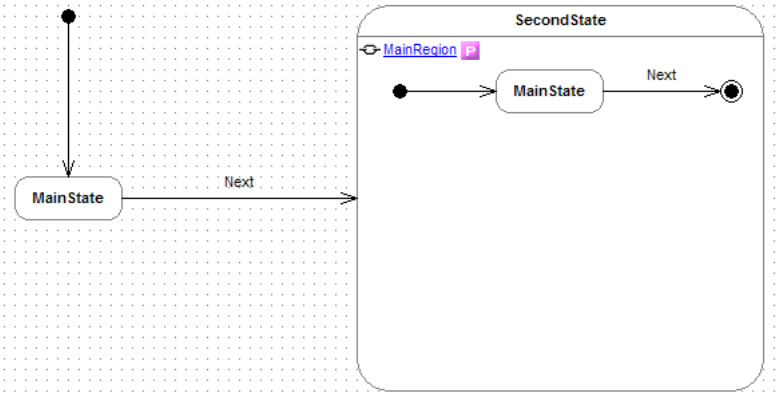
- Guard: This optional OCL expression, if entered, should result in a boolean value. The transition is considered to be available only if this expression is blank or if the expression result evaluates to True.
- Effect: This ActionLanguage expression is executed when the transition is taken, and before the target state is entered.

Transition rules:

- When multiple transitions lead out of a state either all of the transitions must have a trigger or none of them must. There is an exception to this rule when working with composite states (see page 16).
- When multiple transitions leading out of a state have the same trigger you must add a Guard expression to each, where exactly one of those expressions must evaluate as True.

4.3 Composite states

A composite state is a state with one or more state machine diagrams within it. To create a composite state add a standard State to the diagram, and in its property dialog set its State type to "Concurrent Composite". I recommend unselecting "Auto size width" and "Auto size height" so that the state may be manually resized to make space for your embedded diagrams. You will now have a state with a single embedded state machine.

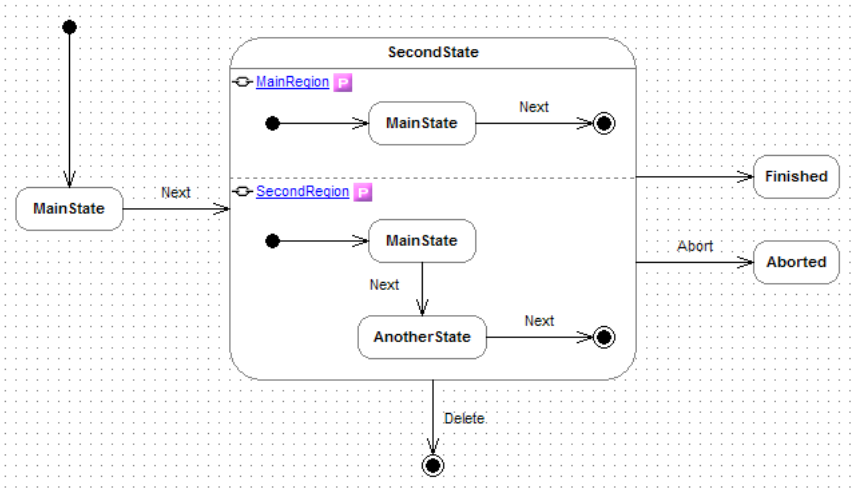


The "P" icon next to the region's name may be clicked to invoke the property dialog for the new region. Within this dialog you may set the following properties:

- Name : This must be a unique region name within the owning state.
- State attribute: This identifies the property (UML attribute) that will hold the state for this region when the instance is persisted. This property should have its "State Attribute" check box checked. A new state attribute property is added to the class automatically whenever you add a new region.

Concurrent states

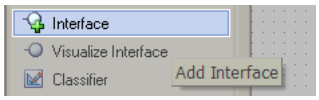
To add an additional concurrent state choose the "Region" item in the tool palette and then click on the client area of the state, a new region will be added. You may now add another state diagram which will execute concurrently alongside the one in the first region.



- In the example diagram the Initial state is located and the transition into MainState is taken immediately.
- When the Next() method on the class is executed the transition into SecondState will be taken.
- Once this composite state is entered both of its concurrent regions will activate.
  - MainRegion will immediately move to MainState.
  - SecondRegion will immediately move to MainState.
  - When Next() is executed the MainRegion will move to its final state and therefore be completed, SecondRegion will move into AnotherState.
  - If Next() were executed again the SecondRegion would also move into its final state, at which point ECO will find a transition with no trigger either with no guard or a guard that evaluates to True and then take that transition; in this case into the "Finished" state.
- If at any point Delete() or Abort() is executed whilst the main state machine is SecondState the correct transition will be taken.

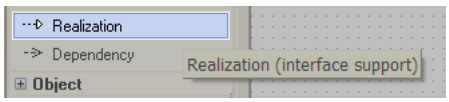
5 Defining interfaces and realization

EcoModeler allows you to both define interfaces and also specify which classes realize them. To add an interface select "Interface" from the tool palette and then click the diagram surface to add it.

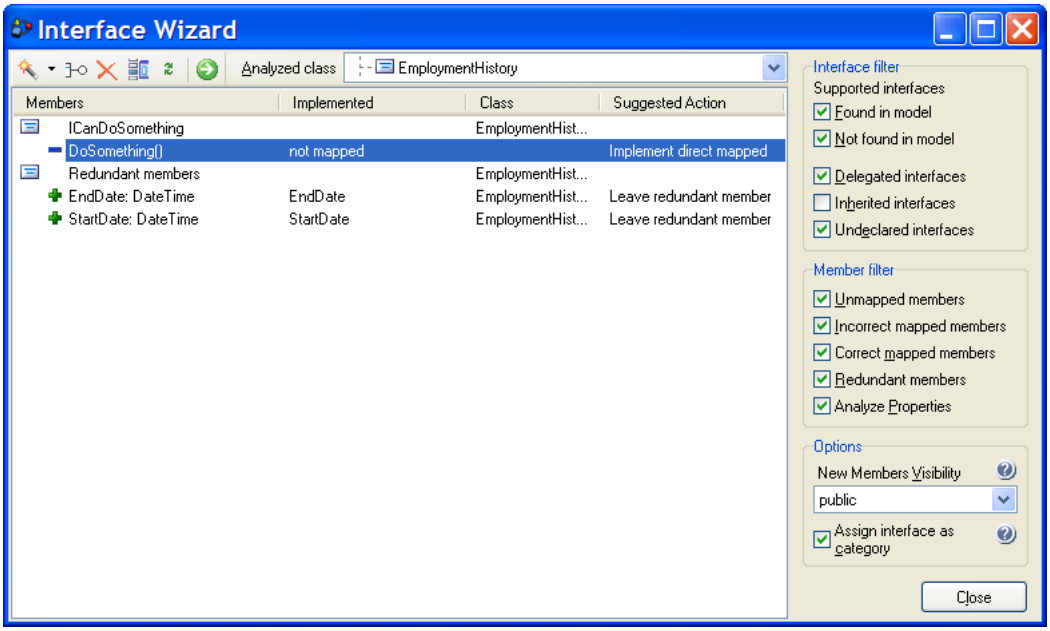


On the dialog that appears give the interface a unique name such as IDoSomething. If you tick the "Placeholder" check box then no source code will be generated for this interface, instead it is assumed to have been defined elsewhere such as in another assembly. If you do not check this box then the interface will generate source code into a single file named ModelInterfaces.cs.

Once you have finished adding methods etc to the interface definition you may wish to get a class to "realize" the interface you have defined. Select the "Realization" item in the tool palette.

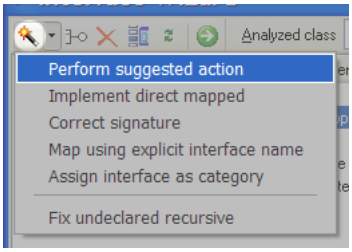


Now click the class which will implement the interface, hold the mouse button down, move the mouse over the interface element on the diagram, and then release the mouse button. At this point a wizard dialog will appear:

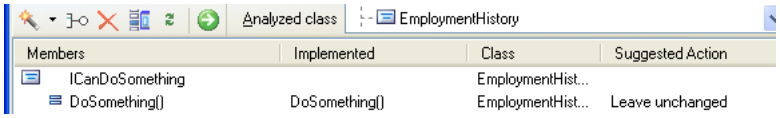


This wizard shows a list of members belonging to the interface first, showing whether or not they are currently mapped to any existing member within the class. Beneath this list there is a list of members that exist within the class that are unrelated

to the interface. To implement the members required to satisfy the interface click the wand button at the top left of the dialog, or click the drop down arrow to the side of it and select "Perform suggested action" from the context menu.



Performing the suggested actions will create the necessary members on the target class:



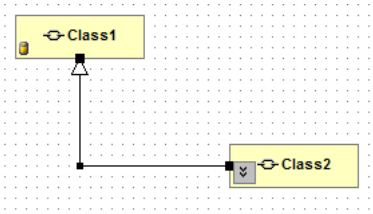
Additionally it is possible to specify which interfaces a class implements (or remove a declaration) by double-clicking the class to bring up its property dialog, and then modifying the comma delimited list in the Interfaces text box.

6 Tidying up diagrams

To improve the appearance of your diagrams there are a number of things you can do.

Way points

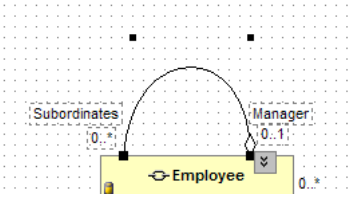
A way point (anchor) is a way of specifying points in a diagram line (associations, transitions, realizations, etc) along which the line must be drawn. This makes it possible to have lines avoid other diagram elements for example.



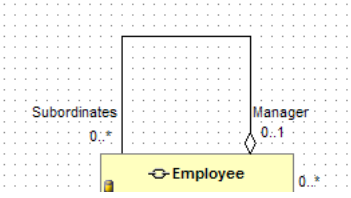
To add an anchor to a line hold down the CTRL key, then click the line at any point and hold the mouse button down. Move the mouse cursor to the desired anchor point and then release the mouse button. To remove an anchor point click the line to reveal all points, then right-click the point in question. On the context menu select "Association" and then "Remove shape node".

Self referencing associations

When your model has an association where both ends are connected to the same class you are able to modify the path of the Bezier curve by moving the anchor points after clicking the line to select it.

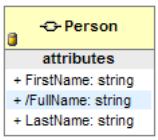


Alternatively you can double-click the line and on the main Association table unselect the "Use Bezier recurrent line style" check box.

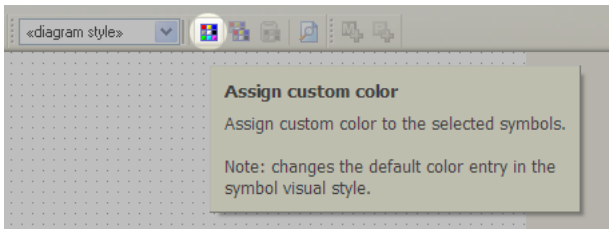


Color

EcoModeler will automatically change the color of properties that have been marked as Derived.



In addition to this you may wish to add color to various other parts of your diagram. For example, you may use Black for persistence associations, Green for transient associations, and Red for derived associations. To change the color of a line on a diagram first select the line by single-clicking it with the mouse, and then select a new color by clicking the "Assign custom color" icon at the top of the diagram surface.



The same operation may be performed on classes too. This is a common practise known as "Modeling in color", see http://en.wikipedia.org/wiki/UML_colors for details.

Default appearance

On the main menu select Options and then Project Options. On the dialog that appears select Diagrams/Classifiers on the tree view on the left hand side. This dialog contains various ways to modify the default appearance for diagrams including which type of members to display (properties / methods, visibility) and also how to arrange the members (member grouping, and member sorting).

These settings may be overridden on individual diagrams by double-clicking a diagram and then selecting the [Classifiers] tab on the dialog that appears. On this dialog you will see a number of settings that have been repeated from the main project settings. In addition to explicitly setting these values you may also specify that you wish to default the value to that specified in the project settings.

7 Advanced modeling techniques

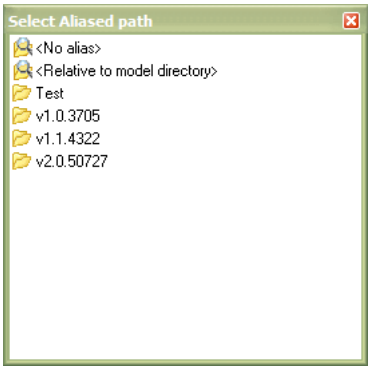
7.1 External references

EcoModeler allows your package to reference classes defined in other packages. This approach allows you to create separate packages with smaller collections of classes and then reuse them. For example you might consider creating packages such as

- CustomerPackage
- EmployeePackage
- OrderingPackage
- StockPackage

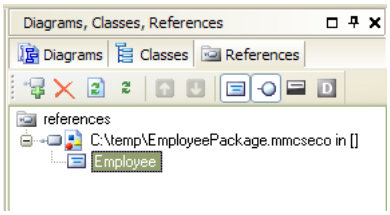
and so on. When a new application is required it is then possible to create a package for your application which references one or more of the classes within these external packages.

1. From the View menu select References.
2. Right-click the References window and select "Add Reference".
3. Next you will be prompted with the following form:



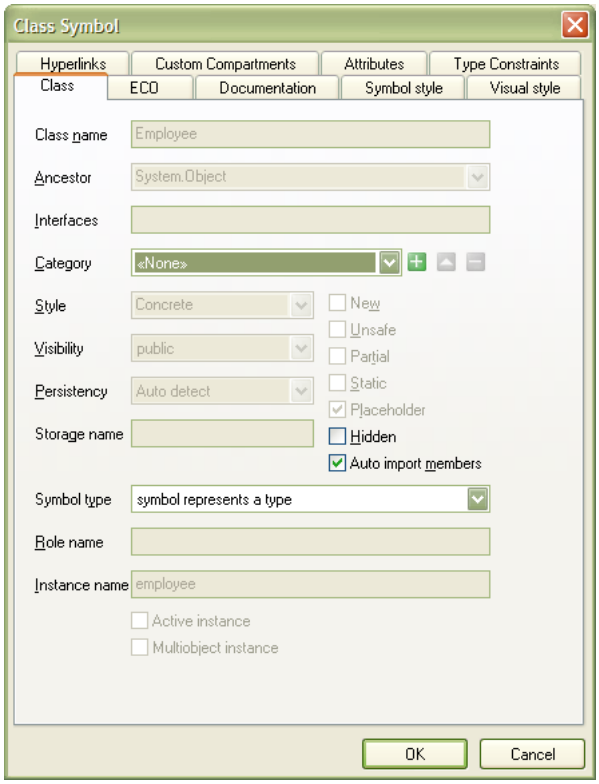
- No alias: This will use a full path to reference the target file.
- Relative to model directory: This will use a path that is relative to the current project.
- Any other value is an alias to a path. These may be defined in the Options->Source Aliases menu.

4. Select the choice most appropriate for your needs, for example the relative option.
5. Now a list of files in that folder will be displayed, click the project file for the package and click OK.
6. Now the selected project will appear in the References tab, expanding the tree view node for that item will reveal the classes in the package.



7. To use one of the referenced classes in your model simply drag it and drop it onto a class diagram.

This will create a "Place holder" class in your project. A place holder is a class which is not actually part of the project but a definition of a class that is known to exist elsewhere. If you double-click the Employee class on your diagram you will see that all of the GUI controls for changing the class are disabled.



Just as the available classes were not automatically added to the project the members of the place holder are not automatically added either. To add the members tick the "Auto import members" check box, you will now see the members of the employee class appear in the class diagram.

It is now possible to descend classes from the place holder class, or add one-way associations to this referenced class. When source code is generated you will need to add a "using ....." (C#) or "uses ....." (Delphi) declaration at the top of your source file, and also ensure that the project in the IDE has a reference to the correct DLL for the referenced class.

7.2 Migrating Together models

It is possible to migrate Delphi ECO models created using Together over to EcoModeler. This is not a fully automated process however, a number a manual steps are involved.

In Delphi it is possible to create multiple packages (*.EcoPkg) within the same project whereas EcoModeler allows only one package in each modeler project file. In Delphi it is common to add multiple packages in order to organize classes and diagrams so that the model is easier to browser. As EcoModeler allows you to add any number of categories for class diagrams this technique is not necessary. As a consequence you must first rearrange your project so that all classes belong to a single ECO Package before importing into EcoModeler.

1. Ensure that you have moved all of your model's classes into a single ECO Package.

Note: This is only necessary for classes within the same model, i.e. will have code generated when you click the "Update ECO source code" button on the [Model view] tab. Any classes in referenced packages need not be changed.

2. Save your Delphi project.
3. In a new EcoModeler project file select the Tools->Convert EcoPackage file menu.
4. Select the EcoPkg file to convert.
5. Click the "Open" button.

The contents of this EcoPkg will now be imported. There are now two manual steps to undertake.

1. State machine diagrams are not imported, you must recreate any diagrams you may have created in your original model.
2. You should generate code to a folder with no source code in it, and then copy any method implementations and custom methods from your original source code and paste them into the empty method stubs in the newly generated code. This is because EcoModeler generates code in a slightly different way than Together and is unable to merge into Together generated code.

Together models with references

To successfully migrate a Together model that has references to external EcoPkg files you must first add suitable references to your model. The referenced file may be either (in order of preference):

- The filename of an EcoModeler project which contains the migrated model information for the referenced Together model.
- The EcoPkg file of the referenced package.
- The DLL produced when the referenced project was compiled. Note that this will also expose the package class as an available class to add to your model, this class should be ignored.

8 Generated source code

Generating source code

Source code is generated or updated by clicking the Generate ECO Model button on the toolbar, by selecting the menu Tools->Generate ECO Model, or by pressing the shortcut key CTRL+F12.

In addition to generating the source code a support file will also be generated named MergeData.ecoxml. This file contains merge information, it enables EcoModeler to find parts of your generated source code so that it can perform operations such as renaming methods etc. **Do not delete or modify this file!** This file should be considered part of your source code, and as such it should be checked in and out of any source control system along with the source code if you intend to generate source code from EcoModeler.

Note: The name of this file has recently been changed. If you have an existing EcoModeler project with source generated you will need to manually rename your existing ecoxml file before you will be permitted to generate source code. This also restricts you to generating no more than one project source into a single folder.

Package class

The name of this class is specified in the menu Options->Project Options->ECO settings. The source code generated for this class has no operational purpose, its purpose is merely to identify which modeled classes belong to the package. The code generator will completely replace this class so you are advised not to add any custom code here at all.

Modeled classes

The source code generated by EcoModeler consists of a number of specially named code folding regions. These regions should not be deleted or renamed, they are used by the EcoModeler code generator to differentiate between locate auto-generated code and manually entered code.

- MM_ECO_Generated : All lines within regions with this name are completely removed and regenerated by the code generator, do not modify this part of the source code.
- MM_ECO AutoGenerated ECO code: This is the same as the MM_ECO_Generated region.
- MM_ECO Model owned attributes: Contains properties for modeled class members such as UML attributes and class associations. Do not remove any properties within this region or change their signatures.
- MM_ECO owned methods: Contains modeled methods and also class constructors. Do not remove any methods within this region or change their signatures.

Example source code

A typical CS source code would look something like the following example. The source has been indented to show the logical regions rather than how a CS file would actually be formatted.

```
using ...;

namespace .....
{
  //First some information on the class
```

```
  #region MM_ECO_Generated
    #region Attributes
      Contains important model information about the class, persistent / transient, OCL
constraints, state machine definition XML, etc.
    #endregion

    #region Documentation
      Only present if you added documentation in the model
    #endregion
  #endregion

  public class MyModeledClass : object, ILoopback {You may add additional interface
declarations here too}
  {
    #region MM_ECO AutoGenerated ECO code
      //Some support code that is of no interest
      ...
      //Sequential indexes for the class's members, useful for
this.AsIObject().Properties.GetByLoopbackIndex(....)
      public struct Eco_LoopbackIndices
      {
        ...
      }
    #endregion

    //Now all modeled UML attributes and associations
    #region MM_ECO Model owned attributes
      //Note: Do not remove any property within this region, nor change its signature

      //An example of a UML attribute
      #region MM_ECO_Generated
        #region Attributes
          Contains important model information
        #endregion

        #region Documentation
          Only present if you added documentation to the model
        #endregion
      #endregion
      public string Description
      {
        get
        {
          //You may add user code here

          #region MM_ECO_Generated
          //Code to get the actual value from the EcoSpace cache, do not write code in
this region.
          #endregion
        }
        set
        {
          //You may add user code here

          #region MM_ECO_Generated
          //Code to set the value in the EcoSpace cache, do not write code in this
region.
          #endregion

          //You may add user code here
        }
      }
    #endregion

    //Now all modeled methods, triggers, and 2 required constructors
    #region MM_ECO Model owned methods

      //Constructor used when recreating the object from the data storage
      public MyModeledClass(IContent content)
      {
        //You may add user code here
```

```
        #region MM_ECO_Generated
        //Do not change code here
        #endregion

        //You may add user code here
      }

      //Constructor used when creating an entirely new instance of a modeled class
      public MyModeledClass (IEcoServiceProvider serviceProvider)
      {
        try
        {
          // Place user code ONLY here
          This is the only part of this method that will survive a subsequent code
generation
        }
        catch
        {
          ..
        }
      }
    #endregion

    You may add non modeled properties / methods etc here, such as interface
implementations and so on.

  }
}
```

9 Deleting model elements

When you delete an element on a class diagram, such as a class or association, you are in fact only removing it from the view and not from the model itself. To clarify; when you see a class on a class diagram you are seeing a diagram element which represents the class in the model and not the class itself, which is why it is possible to show the same class more than once on the same diagram, each showing different properties / methods / associations.

Deleting a class

There are two ways you can delete a class from the model itself (which will remove it from all diagrams.)

1. On a class diagram.
    1. Right-click the diagram element for the class.
    2. On the context menu select Edit->Delete Symbol and Entity.
2. From the Classes tab.
    1. If the classes tab is not visible select the menu View->Classes.
    2. Locate the class in the class hierarchy.
    3. Right-click the class and select Delete.

Deleting a property or method

There is only one way to delete a property or method.

1. Select the class in the Classes tab, or single-click a diagram element representing the class.
2. Ensure the Members view is visible by selecting the menu View->Members.
3. Right-click the property or method in the methods view and select Delete.

Deleting a class association

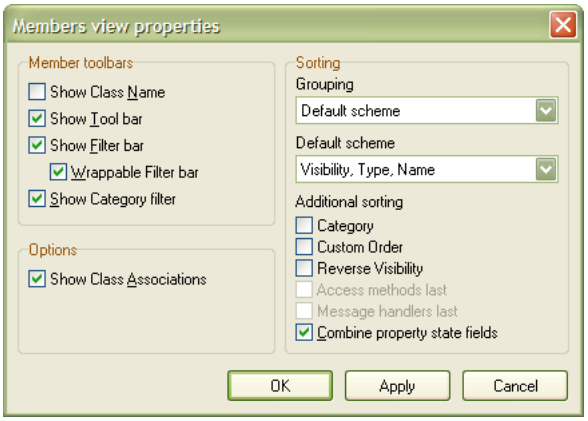
If you wish to change your generated code so that only one of the classes has a reference to the other you need to double-click the association and set only one end to navigable or set one end as non-navigable.

If you wish to delete the association entirely you can either

1. On a class diagram.
    1. Right-click the association line.
    2. On the context menu select Edit->Delete Symbol and Entity.
2. Select the class via a diagram or the Classes tab
    1. Ensure the Members view is visible by selecting the menu View->Members.
    2. Right-click the association in the list at the bottom of the members view.
    3. On the context menu select Delete Association.

Note that if you do not see a separator at the bottom of the members view with your associations in then follow these steps

1. Right-click the members view.
2. On the context menu select the Properties->Members View item.
3. Ensure the Show Class Associations check box is checked, and then click OK.



Deleting a state machine

Unlike other diagram types elements on a state machine diagram do not have any elements within the model, the generated code for the class's state machine is generated entirely from the diagram itself. As a consequence deleting an element on a state machine diagram does in fact delete it entirely. To delete the entire state machine you can right-click the diagram in the Diagrams view and select Delete Diagram, any properties in the class for holding states will need to be deleted manually as they are not deleted along with the diagram.

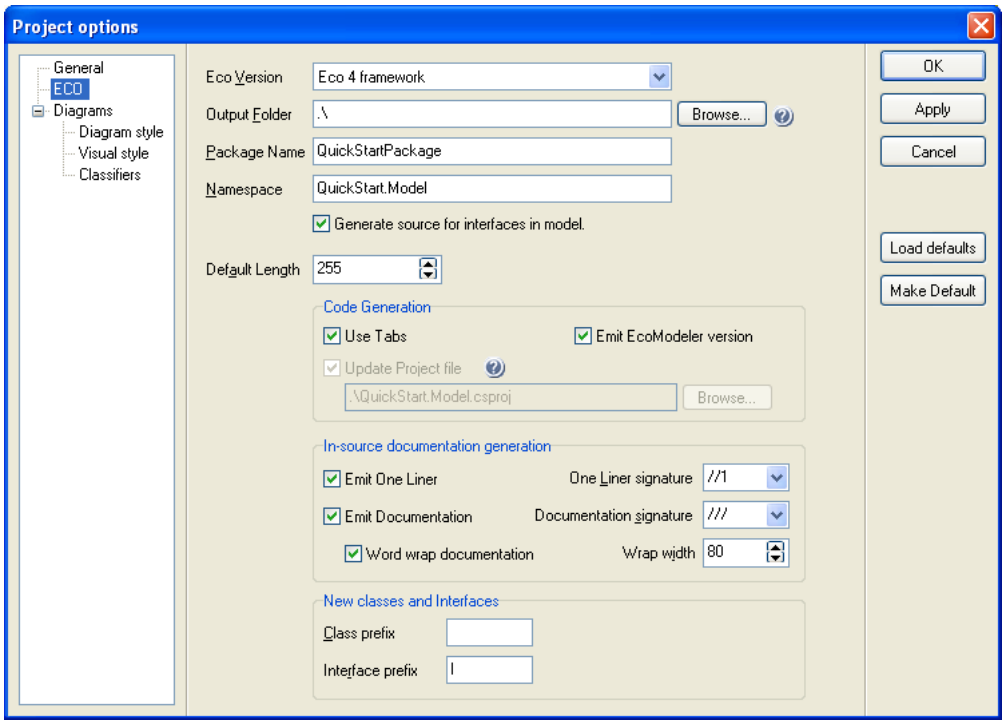
Affects on generated source code

When deleting a property, association, or method the member will be deleted from the generated source. That is unless there is some custom code within it, in which case it will instead be commented out and marked with a TODO reminding you to delete it manually. This is to prevent potentially useful source code from being deleted, just in case you wish to copy/paste the implementation into a new method.

Note: If your intention is to implement the same code using a different name or signature you may simply modify the name etc within the modeler, the next code generation will rename classes / properties etc within your source code.

10 Setting project options

If you want to reconfigure your project options select the *Options* menu, and then the *Project Options* sub menu.



The *Output Folder* specifies where to generate the source code for the modeled business classes. By default the output folder is set to .\, so the source files are generated to the same directory as the .ecommcs file. If you for some reason want to change this, make sure to move all files, including the ModelMergeInfo.ecoxml to the new location.

It is also possible to change the *Package Name* and specify another *Namespace*, it is recommended that you keep the suffix "Package" to the end of your package name. However, If you want to rename the package you need to take the following addtional steps:

1. You may rename the .mmcecocs-file, but it is not required. EcoModeler should be closed when renaming this file.

2. Rename the file PageName.cs in the project. If you don't do this, the file will be removed from the project, and a new one with the new package name created

3. Tools|Generate Eco Model in EcoModeler.

4. Recompile the project (F6)

5. Open the EcoSpace designer in the EcoSpace project, and use the package selector tool to remove the old package name, and add the new one.

When EcoModeler is used with the VS integration the project file is updated automatically when code is generated. Therefore the options for this are grayed out in the version of EcoModeler that ships with ECO for Visual Studio.

- Emit EcoModeler version: This option emits a comment at the top of each generated source file indicating the EcoModeler version number and the date/time the source was generated. Users utilizing a version control system for their source code may wish to disable this option.
- Default Length: This is the length that will be used whenever a new UML attribute (.NET property) is added to a class. As a consequence the Length value on UML attributes is used to determine column sizes when modifying a database structure.
- Update project file: This option should remain checked, and the path to the project file should be specified so that the project is updated whenever the model is modified and code is generated.

Index

A

Adding a class 4

Adding a method 5

Adding a property 6

Adding a state machine diagram 14

Adding association classes 11

Adding class associations 9

Advanced modeling techniques 22

C

Composite states 16

Creating a class diagram 3

D

Defining interfaces and realization 18

Deleting model elements 28

E

External references 22

G

Generated source code 25

Getting started 1

I

Inheritance 8

L

Layout 2

M

Migrating Together models 24

S

Self referencing associations 11

Setting project options 30

States 14

T

Tidying up diagrams 20

Transitions 15