

# **ECO Services**

# Table of Contents

<b>Overview</b>	<b>1</b>
<b>IEcoServiceProvider</b>	<b>1</b>
<b>Terminology</b>	<b>2</b>
<b>Standard ECO services</b>	<b>3</b>
<b>IExternalIdService</b>	<b>4</b>
<b>IStateService</b>	<b>6</b>
<b>IDirtyListService</b>	<b>8</b>
<b>IUndoService</b>	<b>9</b>
StartTransaction, RollbackTransaction, and CommitTransaction	9
Undo blocks	10
Working with the undo service	12
Working with an undo block	15
<b>IObjectFactoryService</b>	<b>16</b>
<b>IVariableFactoryService</b>	<b>18</b>
<b>Query services</b>	<b>22</b>
<b>IOclPsService</b>	<b>25</b>
OCL operations supported by IOclPsService	28
+	28
-	28
*	29
/	29
<	29
<=	30
<>	30
=	30
>	31
>=	31
AllInstances	31
Average	32
Difference	32
Div	32
Exists	33
ForAll	33
Implies	34

---

Includes	34
Intersection	34
IsEmpty	35
IsNull	35
Length	36
MaxValue	36
MinValue	36
Mod	37
Not	37
NotEmpty	38
OrderBy	38
OrderDescending	39
OrderGeneric	39
Reject	40
Select	40
Size	40
SqlLike	41
SqlLikeCaseInsensitive	41
Sum	42
ToLower (String)	42
ToUpper(String)	43
Union	43
IOclService	44
OCL operations supported by IOclService	48
-	49
+	50
AllInstancesAtTime	50
AllLoadedObjects	51
AllSubClasses	51
AllSuperClasses	51
AssociationEnds	52
AsString	52
At	52
AtTime	53
Attributes	53
Collection operations	53
Compare	61
Constraints	62
Create	63
Date and time operations	66
EmptyList	83
Existing	83

---

---

ExternalId	84
If	84
Let	84
Mathematical operations	85
MaxLength	94
ModifiedSinceTimeStamp	94
NewGuid	95
ObjectFromExternalId	95
ObjectTimeStamp	95
OclAsType	96
OclsKindOf	96
OclsTypeOf	97
Parse	97
SafeCast	98
State machine operations	98
String operations	101
SuperTypes	124
TimeSpan operations	124
TaggedValue	124
ToByte	125
ToDouble	125
ToInt16	125
ToInt32	125
ToSByte	126
ToSingle	126
ToUInt16	126
ToUInt32	127
ToUInt64	127
TypeName	127
Xor	127
IActionLanguageService	128
Operations support by IActionLanguageService	128
Clear	128
Add	129
Create	129
Delete	129
Remove	129
RemoveAt	130
ITypeService	130
InstalledOperations	131
Creating a string operation	131
Creating a collection operation	132

---

---

<b>IPersistenceService</b>	<b>133</b>
Multi user concurrency	139
<b>IExtentService</b>	<b>139</b>
<b>IVersionService</b>	<b>141</b>
<b>ICacheContentService</b>	<b>144</b>
<b>Subscriptions</b>	<b>144</b>
<b>ITypeSystemService</b>	<b>147</b>
IModelElement	149
IPackage	149
ITaggedValue	151
IStructuralFeature	151
IClass	152
<b>Registering custom services</b>	<b>159</b>
<b>Replacing standard ECO services</b>	<b>160</b>
Replacing the ExternalIdService	160
A validating IPersistenceService	160
<b>Index</b>	<b>a</b>

---

# 1 Overview

The ECO framework has been designed in such a way that business logic and framework logic are kept as separate as possible. For example, examining the generated source code for an ECO class will not reveal methods such as “Delete” or “Refresh”, as you might expect to find. Keeping framework methods out of our business classes is an important step towards making our source code more readable and manageable. Having a clear and almost invisible separation means that when we inspect the source code of our business classes we only see methods relating to the logical functioning of the class in question. This clearly makes our source code easier to understand, refactor, and debug.

This document will cover two aspects of ECO services. First it will cover the most commonly used services implemented by the ECO framework itself, afterwards it will show how you may create and consume your own services and also show how separating functionality into services can improve your code by separating logic and making it easier to write unit tests.

## 1.1 IEcoServiceProvider

The IEcoServiceProvider is an interface that is realized by the EcoSpace class and is the entry point for retrieving a reference to a previously registered service. The EcoSpace class itself additionally provides a shortcut to the default services, the following code snippet will check if there are currently any class instances that have modified persistent members which need to be saved to the data storage.

```
if (EcoSpace.DirtyList.HasDirtyObjects())  
    ...
```

This same code could be achieved using the IEcoServiceProvider.GetEcoService<T>() generic method. This example is slightly longer than the previous example but illustrates how to retrieve a registered service without relying on the EcoSpace to have a short-cut property, for example when you register your own custom service.

```
if (EcoSpace.GetEcoService<IDirtyListService>().HasDirtyObjects())  
    ...
```

Note: The EcoSpace class itself does not implement IEcoServiceProvider. The interface is actually implemented by another class which the EcoSpace owns an instance of.

### Retrieving services within a business class

ECO business classes are designed so that they may be used in one or more applications, and therefore may be instantiated by one or more different descendants of EcoSpace. When retrieving services from within a business class method it is necessary to go via the IEcoServiceProvider interface.

```
if  
(this.AsIObject().ServiceProvider.GetEcoService<IDirtyListService>().HasDirtyObjects())  
    ...
```

1. **AsIObject()** - This is the entry point to the "ECO world" from a business class. It accesses various support functionality for an object instance within the EcoSpace (they are not implemented by the class itself). The item of interest is the **ServiceProvider** property.
2. **ServiceProvider** - This provides access to the IEcoServiceProvider in order to obtain references to registered services. This reference happens to also be the instantiated EcoSpace which owns the business class instance, however, it is recommended that additional features be implemented via services in order to prevent a strong dependency upon a specific EcoSpace type.

## 1.2 Terminology

As this document discusses both UML models and the source code produced there will be a number of terms which may be used to describe what is essentially the same thing, the term used will depend upon the context. For example, when describing Person.FirstName in the context of UML modeling the term "Attribute" would be used, when in the context of source code the term "Property" will be used. The following table is a list of terms and their meanings

UML model	Source code	Description
Attribute	Property	The term "Attribute" will never be used to refer to the System.Attribute class. System.Attribute will be referred to by its fully qualified name if necessary.
Association end Single role	Property	A reference from one modeled class to another is always done via associations and not attributes. Each association has two ends in UML, and either two or one in source code depending on whether or not the association is navigable in both directions.  A single role is an association end which links to at most one instance.
Multi role	Property	A multi role is an association end which links to a list of instances, allowing it to hold zero to many references.
Element	Object instance Property	This term is used to describe either an instance of a modeled object or one of its property values.

## 2 Standard ECO services

The following services are created and registered automatically whenever you create a new instance of an EcoSpace. These services are an interface to the features that ECO implements as standard such as persistence, in-memory transactions, and so on.

A number of the ECO services have overloaded methods where parameters of the type **IObject** are replaced with a parameter of type **IObjectProvider**. **IObject** is what is known as an "object locator", the way in which ECO references instances of ECO business classes internally. IObject is obtained from an instance of a business class like so

```
IObject objectLocator = person1.AsIObject();
```

whereas the actual object instance is retrieved from the object locator like so

```
Person p = objectLocator.GetValue<Person>();
```

The business class itself implements IObjectProvider, so it then becomes possible to write code in either of the two following ways:

```
//Passing IObject
SomeEcoService.DoSomething(person1.AsIObject());

//Passing the business class instance itself (IObjectProvider)
SomeEcoService.DoSomething(person1);
```

When there are overloads for both parameter types I shall illustrate the IObjectProvider approach only, just to save from having to type **.AsIObject()** needlessly.

Note that when ECO services need to read or modify property values of object instances it does not require the use of reflection. How the values are manipulated depends upon whether or not the model specifies that the property has user code associated with it, additional code which needs to be executed in addition to manipulating the ECO cached values.

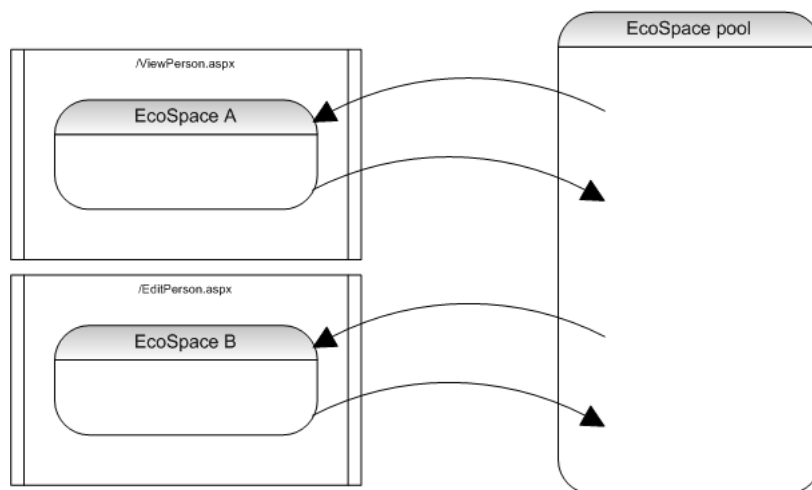
Has user code	Action taken
True	ECO will direct all property access via the object instance itself. To identify which property the service requires some auto-generated code is created in order to get/set property values via an integer index.
False	ECO will access the cache directly, bypassing the object instance completely.



## 2.1 IExternalIdService

Every business class instance within an EcoSpace is uniquely identifiable. Whether this is by an ECO generated object id, or a single/multi part primary key on a database. ECO requires a unique identifier so that it can perform persistence operations on the correct object when updating the database. Using the *IExternalIdService* it is possible to either retrieve a string representation of an object instance's unique identifier, or to provide such a string representation and have ECO provide an object instance, if the object has not already been cached it will first be retrieved from the data storage.

The *ObjectForId()* and *IdForObject()* methods of the *IExternalIdService* may be used to hold weak references to objects, or to pass business class instance references between different EcoSpace instances. This is especially prevalent in ECO powered web service / web application projects where you may wish to use a pool of EcoSpaces and therefore might not always working with the same EcoSpace instance across different page requests. If there is more than one EcoSpace in your web application's/service's EcoSpace pool (which is recommended) then the flow in Figure 05 illustrates a likely scenario.



A user views an object in *ViewPerson.aspx* and then decides to edit that object, at which point they are redirected to *EditPerson.aspx*. Storing the *Person* object in a session is an incorrect approach because the *Person* instance belongs to **EcoSpace A**, whereas *EditPerson.aspx* was allocated **EcoSpace B** from the pool. Instead of holding ECO business class instances between page requests the web application/service should instead hold the "ID" of the object, which is retrieved using *IExternalIdService.IdForObject()*. The receiving page should retrieve an object instance from its allocated EcoSpace using the mirror method *IExternalIdService.ObjectForId()*.

Although this service is used primarily for web applications/web services, there are many more possible applications. Any time the identification of an ECO business class instance needs to be stored in some way this service is the answer. Additionally when developing an ECO WinForms application you may wish to use multiple instances of your application's EcoSpace so that cached data is released as soon as an individual form is closed, the *ExternalIdService* is an excellent way of passing object identities between these forms.

```
//Sending the ID of an object in Form1
string personID = EcoSpace.ExternalIds.IdForObject(person);
SomeOtherForm.EditPerson(personID);
```

```
//Retrieving the object from the ID
Person p = EcoSpace.ExternalIds.ObjectForId(personID).GetValue<Person>();
```

### Object ID structure

The structure of these ID's is always in the format  $\{ClassID\}!\{InstanceID\}$ , an ID identifying the class, followed by an exclamation mark, followed by an ID identifying the instance itself. There are two points at which the ID for an object may change

1. A new instance is persisted
  - Before - \$new\$a791bdee-7cae-4cd6-882d-c983274a65ea!0
  - After - 0!1
2. The model is changed and the application rerun
  - Before - 0!1
  - After - 2!1

Note: The instance ID is an integer by default, but may be another data type such as a Guid depending on how you configure your persistence.

In the first case the ID is constructed by representing the class ID as the string  $\$new\$$  followed by a Guid, and the instance ID as zero. The purpose of this is to prevent the ID of a new instance from being passed to another EcoSpace instance before it is persisted. When passing object references using external ID's the target EcoSpace is able to locate the object either in its cache or by fetching it from the data storage. If the ID being passed is a reference to a new instance then the target EcoSpace instance has no way of locating the object. Once the new instance has been persisted its ID will change to the format in the second example.

In the second example the ID is constructed by representing the class ID as an integer and the instance ID as the key the instance was assigned when it was first persisted to the data storage. The class ID is determined by locating the class in the model's list of classes. The list is sorted so that it remains in a predictable order, however, when the model is changed this can lead to classes appearing in the list or being removed from the list and can therefore cause the class index to change.

External ID's were designed only to be used for short-term references in order to pass instance references between EcoSpace instances. Therefore it is not recommended that these ID's be held onto long term, for example

- Storing the ID in a config file.
- Storing the ID in a persistent property (associations are better suited for this purpose anyway).
- Using the external ID as a parameter in a URL which might later be archived by search engines.

It is possible to replace the ExternalIdService if you wish to ensure that external ID's do not change when the model changes if you require it. This is a technique that will be demonstrated later in this document. See Replacing the ExternalIdService ([see page 160](#)) for details.

## 2.2 IStateService

Whenever a new instance of a business class is created or an existing instance is modified ECO keeps track of these kinds of actions so that it later knows how to update the data storage. The IStateService provides a way of accessing these states.

### bool IsNew(IObjectProvider obj)

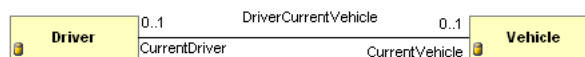
If the ECO business class instance passed has been created and not yet persisted to the data storage this will return true. Once the object has been persisted this method will always return false, even if the object is later modified and requires its changes to be persisted.

### bool IsDirty(IProperty property)

If IsNew() returns true for the object instance that owns the property then this method will always return true. If the member in question has not been modeled as persistent (it is transient or derived) then this method will return false.

If the member has been modeled as persistent and has a change which needs updating to the data storage this method will return true. In the case of simple members such as Int32 and String the result will depend simply on whether or not a change has been made. When the member is an association it is not considered dirty (modified) if it is not embedded. Here are some scenarios which illustrate this last point:

1. Neither end is marked as embedded in the model. An additional table will be created in the database with the name of the association "DriverCurrentVehicle".



```

Driver driver = new Driver(EcoSpace);
Vehicle vehicle = new Vehicle(EcoSpace);

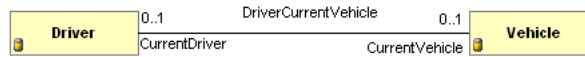
//Get the property references for use with ECO services
IProperty driverCurrentVehicleProp = driver.AsIObject().Properties["CurrentVehicle"];
IProperty vehicleCurrentDriverProp = vehicle.AsIObject().Properties["CurrentDriver"];

//Both return true because the object itself is new
Debug.WriteLine(EcoSpace.States.IsDirty(driverCurrentVehicleProp));
Debug.WriteLine(EcoSpace.States.IsDirty(vehicleCurrentDriverProp));

EcoSpace.UpdateDatabase();
driver.CurrentVehicle = vehicle;

//Both return false because neither end is embedded
Debug.WriteLine(EcoSpace.States.IsDirty(driverCurrentVehicleProp));
Debug.WriteLine(EcoSpace.States.IsDirty(vehicleCurrentDriverProp));
  
```

2. The Driver end of the association is marked as embedded. The Vehicle table in the database will have an additional column named "CurrentDriver" which holds the ID of the driver instance.



```

Driver driver = new Driver(EcoSpace);
Vehicle vehicle = new Vehicle(EcoSpace);

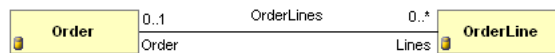
//Get the property references for use with ECO services
IProperty driverCurrentVehicleProp = driver.AsIObject().Properties["CurrentVehicle"];
IProperty vehicleCurrentDriverProp = vehicle.AsIObject().Properties["CurrentDriver"];

//Both return true because the object itself is new
Debug.WriteLine(EcoSpace.States.IsDirty(driverCurrentVehicleProp));
Debug.WriteLine(EcoSpace.States.IsDirty(vehicleCurrentDriverProp));

EcoSpace.UpdateDatabase();
driver.CurrentVehicle = vehicle;

//Returns false because it is not embedded
Debug.WriteLine(EcoSpace.States.IsDirty(driverCurrentVehicleProp));
//Returns true because the ID of the Driver is embedded into this class's table
Debug.WriteLine(EcoSpace.States.IsDirty(vehicleCurrentDriverProp));
  
```

3. The Order end of the association is marked as embedded. The OrderLine table in the database will hold the column referencing which order it belongs to.



```

Order order = new Order(EcoSpace);
OrderLine orderLine = new OrderLine(EcoSpace);

//Get the property references for use with ECO services
IProperty orderLinesProp = order.AsIObject().Properties["Lines"];
IProperty orderLineOrderProp = orderLine.AsIObject().Properties["Order"];

//Both return true because the object itself is new
Debug.WriteLine(EcoSpace.States.IsDirty(orderLinesProp));
Debug.WriteLine(EcoSpace.States.IsDirty(orderLineOrderProp));

EcoSpace.UpdateDatabase();
order.Lines.Add(orderLine);

//Returns false because it is not embedded
Debug.WriteLine(EcoSpace.States.IsDirty(orderLinesProp));
//Returns true because the ID of the Order is embedded into this class's table
Debug.WriteLine(EcoSpace.States.IsDirty(orderLineOrderProp));
  
```

4. Neither end of this association may be embedded because both ends are multiple (not all databases support multi-values in a single column). An additional table is created in the database with the name of the association (FoodLikedBy) in order to maintain the associations.



```

Person wallis = new Person(EcoSpace);
Food cheese = new Food(EcoSpace);

//Get the property references for use with ECO services
  
```

```

IProperty personFoodLikedProp = wallis.AsIOBJECT().Properties["FoodLiked"];
IProperty foodLikedByProp = cheese.AsIOBJECT().Properties["LikedBy"];

//Both return true because the object itself is new
Debug.WriteLine(EcoSpace.States.IsDirty(personFoodLikedProp));
Debug.WriteLine(EcoSpace.States.IsDirty(foodLikedByProp));

EcoSpace.UpdateDatabase();
wallis.FoodLiked.Add(cheese);

//Returns false because neither end may be embedded
Debug.WriteLine(EcoSpace.States.IsDirty(personFoodLikedProp));
Debug.WriteLine(EcoSpace.States.IsDirty(foodLikedByProp));

```

### bool IsDirty(IOBJECTProvider obj)

If any of the properties for the passed object instance are considered dirty or the object instance is new or marked as deleted then this method will return true, otherwise there are no changes to persist to the data storage and this method will return false.

```

Person person = new Person(EcoSpace);
//Will return true because the object is new
Debug.WriteLine(EcoSpace.States.IsDirty(person));

EcoSpace.UpdateDatabase();
//Will return false, the object remains unaltered since it was persisted
Debug.WriteLine(EcoSpace.States.IsDirty(person));

person.AsIOBJECT().Delete();
//Will return true, the object has not been removed from the data storage since deleted
Debug.WriteLine(EcoSpace.States.IsDirty(person));

EcoSpace.UpdateDatabase();
//Will return false, the object cannot be dirty now that it no longer exists
Debug.WriteLine(EcoSpace.States.IsDirty(person));

```

## 2.3 IDirtyListService

Whenever a persistent object is created, modified, or deleted, it is considered to be "Dirty"; this means that it has in some way been altered. This Dirty state is an indication that the data storage needs to be updated in order to reflect the changes made to the object instance in question.

Using the IDirtyListService interface the dirty list service enables the developer to obtain a list of objects that have been modified. Note that the "Dirty" state is reserved for persistent objects only, object instances of a class marked as Transient in the model are never saved to the persistence storage and therefore cannot have such a state. Additionally, if the EcoSpace has no persistence defined then all classes are considered to be Transient regardless of how they have been defined in the model, in such a case the IDirtyListService will never hold any object references.

### IOBJECTList AllDirtyOBJECTs()

This method returns an IOBJECTList containing an IOBJECT for each dirty object held within the EcoSpace. This list is immutable, meaning that if you try to modify it using Remove() for example a System.InvalidOperationException will be thrown. The size of this list will alter as more objects become dirty, the data storage is updated marking some objects as non-dirty, or as a result of objects becoming dirty/non-dirty due to in-memory transactions being rolled back or reapplied -

see IUndoService (see page 9).

### bool HasDirtyObjects()

If *AllDirtyObjects().Count* is greater than zero then this method will return true, otherwise it will return false indicating that there are no dirty objects to persist to the data storage.

### void Subscribe(ISubscriber subscribe)

Passing an instance of ISubscriber to this method will cause the *ISubscriber.Receive* method to be executed whenever the number of objects held in the *AllDirtyObjects()* list alters. This is a useful global hook to perform tasks on objects that have been modified, for example to provide on-screen validation.

---

## 2.4 IUndoService

Using the IUndoService the programmer is able to perform in-memory transactions on objects within the EcoSpace. These transactions may be committed, reversed, or reapplied at any point ensuring that if an operation fails the state of all affected objects is returned to a specific state.

To use a classic example; if a funds transfer is initiated from bank account "A" to bank account "B" two operations must take place. The balance of account "A" must decrease by the transaction value and the balance of account "B" must increase by the transaction value. This kind of atomic operation has been available in all good databases for quite some time now, but the ECO undo service allows the same kind of atomic operation to be performed in-memory as well, eliminating the need to reload object contents from the data storage if you wish to abandon a set of changes.

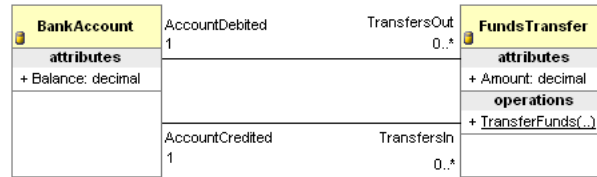
The undo service provides two main pieces of functionality. Firstly it provides named undo-blocks; changes within the undo block may be reversed or reapplied repeatedly. Secondly it provides in-memory transaction support, which internally uses the undo-block functionality to provide nested *StartTransaction*, *RollbackTransaction*, and *CommitTransaction* methods.

The undo blocks are also affected by the persistence service (see page 133) whenever an update is performed.

---

### 2.4.1 StartTransaction, RollbackTransaction, and CommitTransaction

This first example will demonstrate how to perform an in-memory transaction on a number of objects within the EcoSpace. The example will transfer a given amount of money from one bank account to another, it will adjust the *CurrentBalance* of each account and additionally create a transaction object to record the transfer. If an exception of some kind occurs within the transfer method then all changes will be rolled back, otherwise the in-memory transaction will be committed.



```

public class FundsTransfer .....
{
    ...
    public static void TransferFunds(IEcoServiceProvider serviceProvider, BankAccount
debitAccount, BankAccount creditAccount, decimal amount)
    {
        //Perform parameter validation here
        IUndoService undoService = serviceProvider.GetEcoService<IUndoService>();
        undoService.StartTransaction();
        try
        {
            FundsTransfer transfer = new FundsTransfer(serviceProvider);
            transfer.AccountDebited = debitAccount;
            transfer.AccountCredited = creditAccount;
            transfer.Amount = amount;
            debitAccount.Balance -= amount;
            creditAccount.Balance += amount;

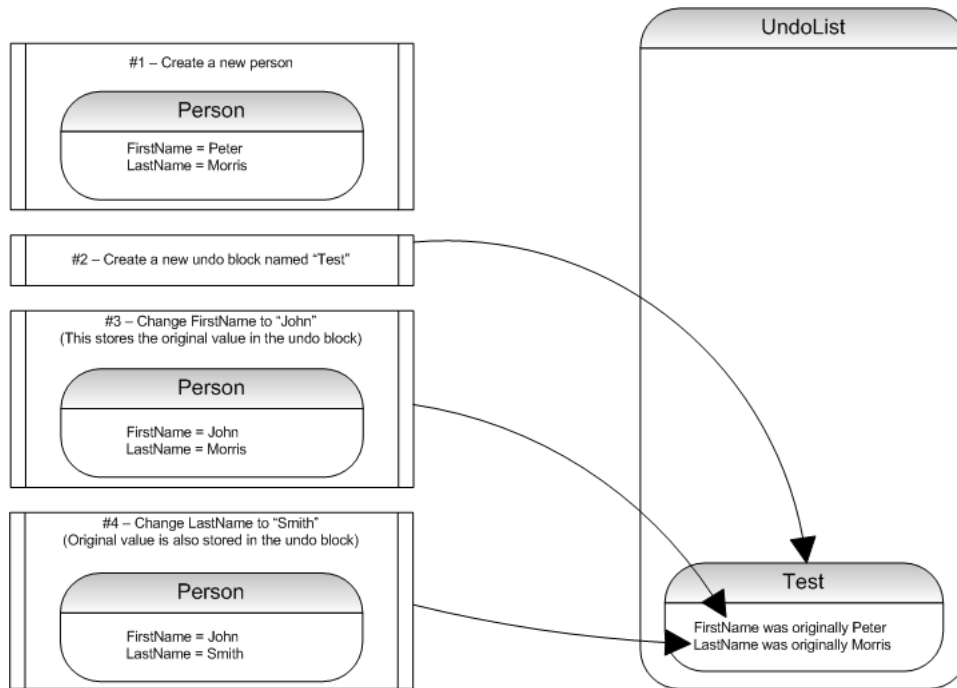
            //Perform overdraft validation etc here
        }
        catch
        {
            undoService.RollbackTransaction();
            throw;
        }
        undoService.CommitTransaction();
    }
}
  
```

1. A reference to the IUndoService is obtained.
2. An in-memory transaction is started.
3. A new FundsTransfer object is created to record the transfer.
4. The FundsTransfer object is associated with the debit and credit accounts.
5. The balances of both the credit and debit accounts are modified.

If all goes as expected the EcoSpace will have a new FundsTransfer object to persist to the data storage along with modified account balances. If any exception is thrown during the operation the in-memory transaction is rolled back restoring the EcoSpace to its former state, the FundsTransfer object will no longer exist and the two accounts in question will remain unaltered.

## 2.4.2 Undo blocks

Undo blocks provide a mechanism similar to transactions, in fact the transaction mechanism internally uses undo blocks. Whenever a modification is made to an ECO element (object / property) ECO will check if there is an active undo block. If an undo block is found, and the element in question is not already in the active undo block, ECO will record the elements original value in the undo block. In the case of a property the original value will be recorded, in the case of an IObject its state will be recorded (new, existing, deleted).



Holding a collection of elements plus their original states allows the changes recorded in an undo block to be reversed, restoring the `EcoSpace` to the exact state it was in at the point the undo block was created. The undo service may hold multiple undo blocks, only the topmost undo block is considered to be active therefore changes made within the `EcoSpace` will always be applied only to the topmost undo block, new undo blocks are always placed at the top of the undo list. This makes it possible to have multiple separate transactions being performed within the `EcoSpace` at the same time, each with the ability to be independently reversed or reapplied.

One example use of the undo service is when working with multiple forms against a single `EcoSpace` instance. Each form could own its own undo block and move it to the top of the list (thus making it active) whenever that form is focused. This would make it possible to track changes made by an individual form and then undo / redo those changes or update the data storage with those changes only. Note that it is recommended to have one `EcoSpace` instance per form whenever possible.

### Creating an undo block

Undo blocks in ECO are identified using a unique block name. Although it is possible to hold a reference to an undo block using an `IUndoBlock` it is only advisable to hold such a reference for a short period of time; only as a local variable for example. The reasoning is quite simple, undo blocks may be removed from the undo service completely (effectively "committed"), accessing the undo block by name will correctly return null whereas holding onto an `IUndoBlock` reference would result in your application performing operations on an undo block that is no longer valid.

To ensure that block names are unique, the undo service provides the `GetUniqueBlockName` method. Executing this method will provide your application with a block name that is guaranteed to be unique.

```
int i = 1;
while (i++ <= 3)
{
    string uniqueName = EcoSpace.Undo.GetUniqueBlockName("Test");
    EcoSpace.Undo.StartUndoBlock(uniqueName);
    MessageBox.Show(uniqueName);
}
```



In this example the application creates three undo blocks. Rather than hard-coding the block name as "Test" the application asks the undo service to return a unique name using "Test" only as a suggested name. The output of the program as each iteration of the loop is executed is as follows; *Test*, *Test1*, *Test2*.

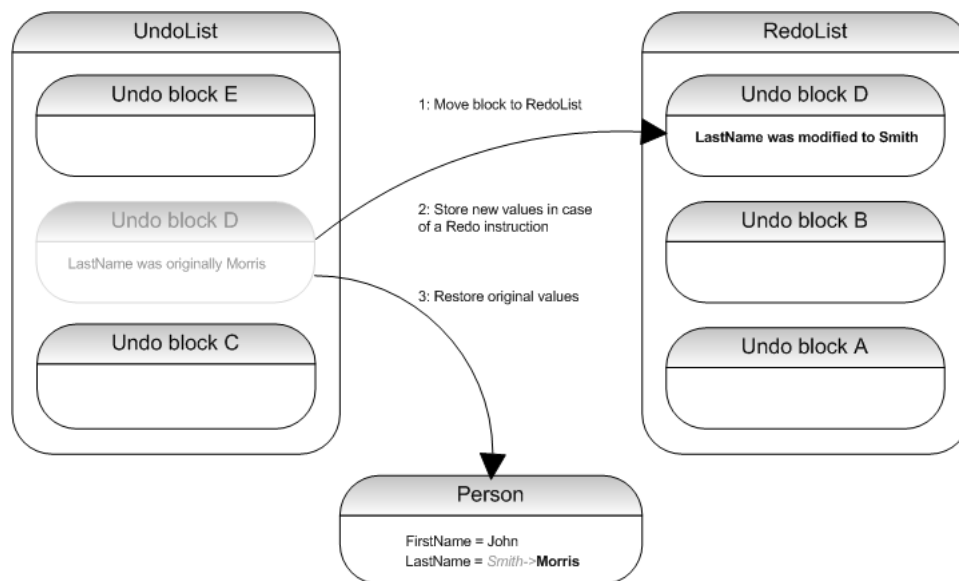
Note that if the application makes a modification to an object in the EcoSpace and there are no undo blocks present ECO will automatically create an undo block named "Unnamed", trying to create an undo block with a name that is already in use will result in a `System.InvalidOperationException` being thrown.

## 2.4.3 Working with the undo service

### Undo operations

Using the `UndoBlock` method it is possible to reverse any changes that have been made since the undo block was activated; i.e. when it became the undo block at the top of the `UndoList`. Calling `UndoBlock` will perform the following actions

1. Move the block into the `RedoList`
2. Store the new values instead of the original values, so that the changes may be reapplied later if necessary
3. Restore the original values. This includes modified members, and also created / deleted object instances



```

Person person1 = new Person(EcoSpace);
person1.FirstName = "Peter";
person1.LastName = "Morris";

EcoSpace.Undo.StartUndoBlock("D");
//Update undo block D with LastName=Morris
person1.LastName = "Smith";

//Restore LastName to Morris
//Record modified LastName as Smith
//Move block to RedoList
EcoSpace.Undo.UndoBlock("D");

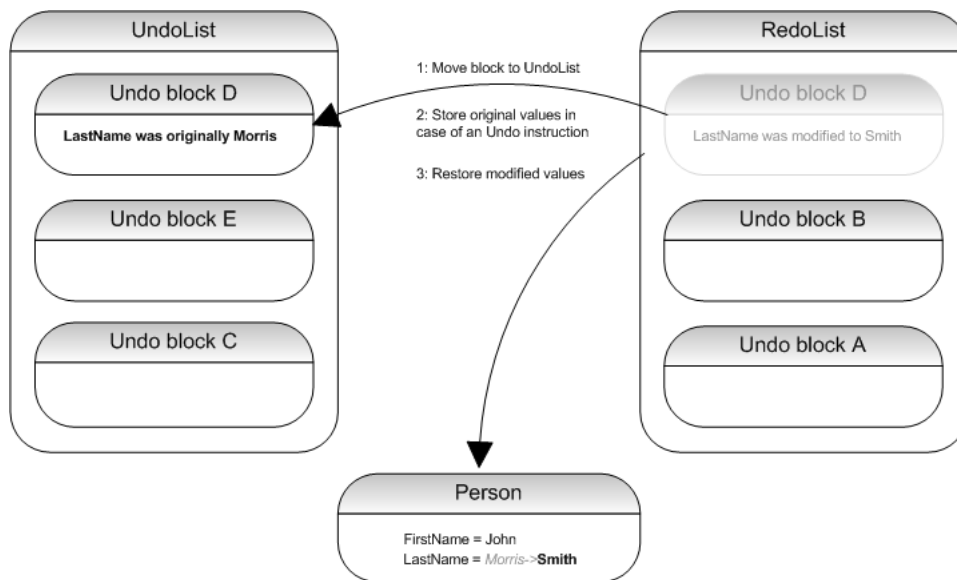
```

You may also use the *UndoLatest* method to undo the block at the top of the UndoList.

### Redo operations

Once a block is in the RedoList it is possible to reapply its changes using the *RedoBlock* method. Whereas an undo operation reinstates the original state a redo operation will reinstates the modifications made. Calling *RedoBlock* will perform the following actions

1. Move the block back to the UndoList
2. Store the very original values in case *UndoBlock* is executed again
3. Restore the modified values



```
#region Code from undo example
Person person1 = new Person(EcoSpace);
person1.FirstName = "Peter";
person1.LastName = "Morris";

EcoSpace.Undo.StartUndoBlock("D");
//Update undo block D with LastName=Morris
person1.LastName = "Smith";

//Restore LastName to Morris
//Record modified LastName as Smith
//Move block to RedoList
EcoSpace.Undo.UndoBlock("D");
#endregion

//Restore modified value LastName=Smith
//Record original value LastName=Morris
//Move block back to UndoList
EcoSpace.Undo.RedoBlock("D");
```

You may also use the *RedoLatest* method to redo the block at the top of the RedoList.

## Moving blocks

Blocks in the UndoList/RedoList may be rearranged. The purpose of this is to bring a different undo block to the top of the list in order to make it active. As mentioned earlier this is useful for example if you wish to track changes made by the user on a form by form basis so that they may be undone/redone independently of each other. Both the UndoList and RedoList implement a `MoveBlock(int currentIndex, int newIndex)` method.

There is a restriction which must be adhered to when moving blocks. When multiple blocks A,B,C contain information about the same element it makes sense logically to undo changes only in the order C,B,A and to redo changes only in the order A,B,C. To enforce this rule ECO will always attempt to move the specified block to the top of its list before either undoing or redoing its contents. If any blocks above the block being moved contain any common elements the process is aborted with a `System.InvalidOperationException`. The following example illustrates this point

```
EcoSpace.Undo.StartUndoBlock("A");
Person person1 = new Person(EcoSpace);
person1.FirstName = "Peter";

//Create a new block at the top of the list
EcoSpace.Undo.StartUndoBlock("B");
person1.FirstName = "John";

//Uncreate person1 - this cannot be done due to undo block B
EcoSpace.Undo.UndoBlock("A");

//Revert person1.FirstName to "Peter"
EcoSpace.Undo.UndoBlock("B");
```

In this example it is not possible to undo the changes in block A first because block B is above it and contains a common element. If this were permitted it would result in block B remaining in the list holding undo information about an object that no longer exists. To determine whether or not it is possible to move a block to a specific location use the `CanMoveBlock(int currentIndex, int newIndex)` method of either the UndoList or RedoList. If your intention is to undo or redo a block rather than to simply move it then you can use either `CanRedoBlock(string blockName)` or `CanUndoBlock(string blockName)`.

To move a block to the top of the UndoList/RedoList you may use the lists' `MoveToTop(string blockName)` methods.

## Merging blocks

Both the UndoList and RedoList contain a `MergeBlocks(string destinationBlockName, string sourceBlockName)` method. When two blocks are merged the information stored in the source block is added to the destination block and then the source block is removed from the list. If both blocks contain information one of these pieces of information must take priority. The following table describes how this priority is determined

List used	Action taken
UndoList	The oldest information takes priority, the change that occurred latest is discarded.
RedoList	The newest information takes priority, the change that occurred earliest is discarded.

As with moving blocks there is a similar restriction. When two blocks are merged ECO will check the contents of each block between them, if any of those blocks contain an element common to either the source or destination block then the merge will not be permitted. The method `CanMergeBlock(int currentIndex, int newIndex)` will indicate whether the merging of two undo blocks is possible.

### Removing blocks

Using *UndoBlock* and *RedoBlock* removes the undo block from its owning list but also inserts it into the opposite list. There are two ways in which an undo block is removed from a block list permanently, without being moved to another list.

1. The data storage is updated - Any block containing changes for any of the objects updated to the data storage will be removed automatically.
2. Manual removal - Executing a list's *RemoveBlock(string blockName)* method will remove the block with the specified name, executing *ClearAllUndoBlocks* will clear both the UndoList and the RedoList.

Once a block has been removed it cannot be manually re-added to the undo mechanism.

## 2.4.4 Working with an undo block

Undo blocks expose a limited number of features via the IUndoBlock interface.

### Retrieving an undo block

An IUndoBlock reference may be obtained from either the UndoList or the RedoList either by name or index.

```
private void button1_Click(object sender, EventArgs e)
{
    EcoSpace.Undo.StartUndoBlock("In UndoList");

    //Create a block and move it to the RedoList
    EcoSpace.Undo.StartUndoBlock("In RedoList");
    EcoSpace.Undo.UndoLatest();

    //Show by name
    ShowBlockName(EcoSpace.Undo.UndoList["In UndoList"]);
    ShowBlockName(EcoSpace.Undo.RedoList["In RedoList"]);

    //Show by index
    ShowBlockName(EcoSpace.Undo.UndoList[0]);
    ShowBlockName(EcoSpace.Undo.RedoList[0]);

    //Non-existent by name
    ShowBlockName(EcoSpace.Undo.RedoList["This does not exist"]);

    //Non-existent by index
    ShowBlockName(EcoSpace.Undo.RedoList[99]);
}

private void ShowBlockName(IUndoBlock undoBlock)
{
    if (undoBlock == null)
        MessageBox.Show("<null>");
    else
        MessageBox.Show(undoBlock.Name);
}
```

Retrieving an undo block using an invalid name or index will return null.

**Members of IUndoBlock**

- Name - The name give to the undo block.
- ContainsChanges - Returns false if the undo block is empty, otherwise returns true.
- GetChangedObjects() - Returns an IObjectList. Each IObject is the identity of the object instance that has been modified.
- Subscribe(ISubscriber subscriber) - The subscriber is notified whenever the contents of the undo block change, when the undo block moves between the RedoList and UndoList, or the when undo block is removed from the undo service altogether.

---

## 2.5 IObjectFactoryService

This service provides an alternative way of creating instances of modeled classes. Ordinarily a new instance of a modeled class is created in an application like so

```
//EcoSpace implements IEcoServiceProvider, so we can pass the EcoSpace
OrderLine newLine = new OrderLine(EcoSpace);
CurrentOrder.Lines.Add(newLine);
```

or to create an instance from a method of another modeled class like so

```
//Business classes don't have an EcoSpace, so we pass the IEcoServiceProvider
OrderLine newLine = new OrderLine(this.AsIObject().ServiceProvider);
this.Lines.Add(newLine);
```

Creating a new object instance is so trivial that it may seem unnecessary to have a service for this purpose, however, the object factory service makes it easy to create instances of modeled classes when the type is not know until runtime, or is determined by reading model information.

**Creating an object instance**

The first approach is the one most similar to the previous example. An instance will be created by passing the System.Type.

```
IObjectInstance instance = EcoSpace.ObjectFactory.CreateNewObject(typeof(Person));
Person person1 = instance.GetValue<Person>();
```

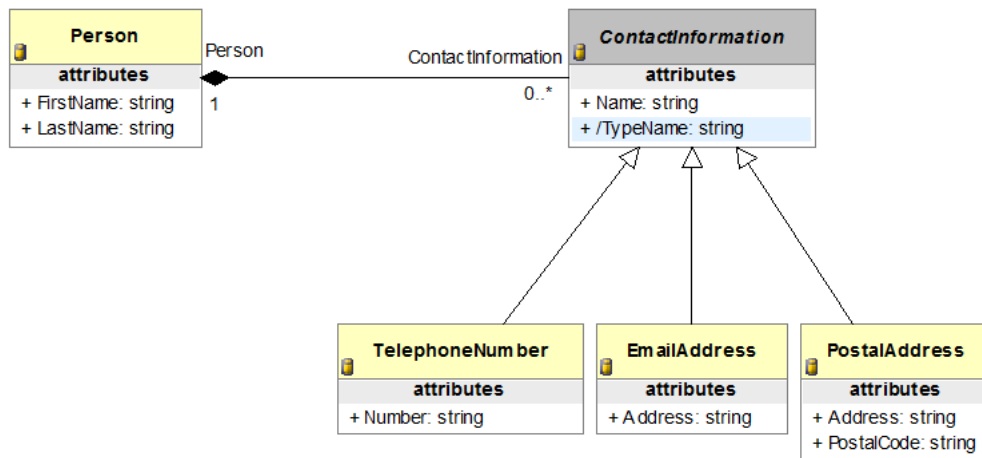
First an IObjectInstance is created by instructing the object factory service to create a type of Person. Next the Person instanced is retrieve from the IObjectInstance object locator. Note that the IObjectInstance reference may be used as a parameter for various other service methods; it is also possible to use person1.AsIObject() to retrieve the IObjectInstance.

The second approach is to use a string to identify the name of the class to create. Unlike the previous approach this approach will not cause a compile error if you change the name of the Person class.

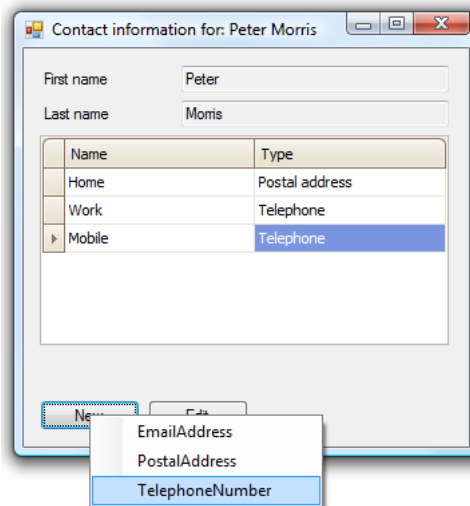
```
IObjectInstance instance = EcoSpace.ObjectFactory.CreateNewObject("Person");
```

The final approach creates an instance identified by an IClass. An IClass is an interface holding information about a specific

class in the model, this instance holds additional information regarding persistency etc. This is covered in the type system service (see page 147) section of this document.



In the model above the **Person** class may now contain multiple pieces of contact information. The **ContactInformation** class itself is abstract so cannot be instantiated, the subclasses **TelephoneNumber**, **EmailAddress**, and **PostalCode** are all concrete classes and therefore may be instantiated and associated with a person. When the user interface offers the user the opportunity to add some contact information for a **Person** it would be nice if the possible types were determined automatically rather than having to hard code them and have to remember to update the code each time a new kind of contact information was added to the model. This is where the **IClass** comes in. The following code example will show how to perform this task; it will use some code from a service which has not yet been covered so feel free to skip directly to the point in the example where the instance is created.



```

private void ButtonNewContactInformation_Click(object sender, EventArgs e)
{
    //Clear the context menu
    ContextCreateContactInformation.Items.Clear();

    //Find the base IClass
    IClass contactInformationClass =
    EcoSpace.TypeSystem.GetClassByType(typeof(ContactInformation));

    //Recursively add all types
    AddClassToMenu(contactInformationClass);
}
  
```

```

    //Show the menu
    Point popupPoint = new Point(ButtonNewContactInformation.Width / 2,
    ButtonNewContactInformation.Height / 2);
    ContextCreateContactInformation.Show(ButtonNewContactInformation, popupPoint);
}

private void AddClassToMenu(IClass contactInformationClass)
{
    //If this is not abstract then create a menu item
    if (!contactInformationClass.IsAbstract)
    {
        var menuItem = new ToolStripMenuItem(contactInformationClass.Name);
        ContextCreateContactInformation.Items.Add(menuItem);
        menuItem.Tag = contactInformationClass;
        menuItem.Click += MenuItemCreateClass_Click;
    }

    //Now recursively add any sub classes
    foreach (IClass subClass in contactInformationClass.SubTypes)
        AddClassToMenu(subClass);
}

private void MenuItemCreateClass_Click(object sender, EventArgs e)
{
    //Normally we would pass the type to a specific form to edit
    //but for this example we will just add directly to the list
    ToolStripMenuItem menuItem = (ToolStripMenuItem)sender;
    IClass contactInfoClass = (IClass)menuItem.Tag;

    //Create the new instance based on IClass
    IObjectInstance instance = EcoSpace.ObjectFactory.CreateNewObject(contactInfoClass);
    ContactInformation newContactInfo = instance.GetValue<ContactInformation>();
    CurrentPerson.ContactInformation.Add(newContactInfo);
}

```

## 2.6 IVariableFactoryService

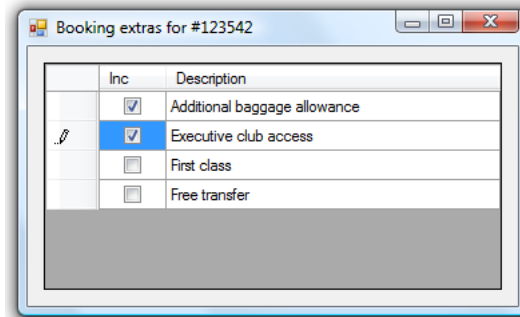
This service enables the developer to create a number of IElement based objects which may then be used in various different parts of the ECO framework. The IElement interface is prevalent throughout the ECO framework as it is used to represent instances of modeled classes (IObject ultimately descends from IElement), property values (Int32, String, etc), and association ends (IObjectList). These variables may then be used in various other services, for example the OCL Service may evaluate OCL expressions which include variable names (a bit like parameterized queries). Although in most circumstances it is anticipated that the developer will want to use a handle for declaring variables this service allows you to create variables in code, which is useful in methods of modeled classes where you have no design surface onto which to drop a VariableHandle etc.

### Creating a constant

A constant is most useful when adding an event derived column to an ECO handle. One very useful implementation of this functionality is when you have a many to many association between two classes and wish to show the user a selection of options with a check box, checking the check box will add the item to the association whereas unchecking the check box will remove the option from the association.



The model above allows a single instance of BookingOption to be associated with many bookings, and for a booking to have many BookingOption instances (extras). The following user interface displays a booking form with a list of all available BookingOption instances rather than only the ones associated with the booking in question.



1. An expression handle was added with the expression "BookingOption.allInstances".
2. The columns were edited and a new EventDerivedColumn was added using the drop down list on the Add button.
3. The DeriveValue and ReverseDeriveValue events on this expression handle were implemented like so

```

private void ehBookingOptions_DeriveValue(object sender, DeriveEventArgs e)
{
    //Get the BookingOption in question
    BookingOption option = e.RootElement.GetValue<BookingOption>();
    switch (e.Name)
    {
        case "Included":
            //Result is true if this option is in CurrentBooking.Extras
            bool isIncluded = CurrentBooking.Extras.Contains(option);
            //Create the boolean constant and specify it as the result
            e.ResultElement = EcoSpace.VariableFactory.CreateConstant(isIncluded);

            //Additional: We need to tell ECO to update the ticks in the boxes whenever
            //the size of CurrentBooking.Extras changes
            //01: Get the IProperty for CurrentBooking.Extras
            int propIndex = Booking.Eco_LoopbackIndices.Extras_MemberIndex;
            IProperty extrasProperty =
                CurrentBooking.AsIObject().Properties.GetByLoopbackIndex(propIndex);
            //02: Subscribe to it
            extrasProperty.SubscribeToValue(e.ResubscribeSubscriber);
            break;

        default:
            throw new NotImplementedException(e.Name);
    }
}

private void ehBookingOptions_ReverseDeriveValue(object sender, ReverseDeriveEventArgs e)
{
    //Get the BookingOption in question
    BookingOption option = e.RootElement.GetValue<BookingOption>();
    switch (e.Name)
    {
        case "Included":
  
```



```

    bool isIncluded = (bool)e.Value;
    //Add or remove it from the list. This will update the
    //UI automatically because when we derived the value we
    //subscribed to CurrentBooking.Extras
    if (isIncluded)
        CurrentBooking.Extras.Add(option);
    else
        CurrentBooking.Extras.Remove(option);
    break;

default:
    throw new NotImplementedException(e.Name);
}
}

```

### Creating a list of objects

When creating an `IObjectList` you have the option of either allowing `IObject` instances representing any modeled class, or ones which represent a specific modeled class and its subclasses only. To create an untyped object list you would use the `CreateUntypedObjectList(bool allowDuplicates)` method, this kind of list is useful when you wish to update the data storage with a collection of objects rather than updating all dirty objects.

```

//Create three people
Person person1 = new Person(EcoSpace);
Person person2 = new Person(EcoSpace);
Person person3 = new Person(EcoSpace);

//Create the object list
IVariableFactoryService vfs = EcoSpace.VariableFactory;
IObjectList objectsToPersist = vfs.CreateUntypedObjectList(false);

//Add only the locators for person1 and person3
objectsToPersist.Add(person1.AsIObject());
objectsToPersist.Add(person3.AsIObject());

//Now update the data storage with only those two objects
EcoSpace.Persistence.UpdateDatabaseWithList(objectsToPersist);

```

To create a typed object list you would use either

- `CreateTypedObjectList(Type type, bool allowDuplicates)`
- `CreateTypedObjectList(IClass umlClass, bool allowDuplicates)`

The first accepts a .NET Type whereas the second accepts an `IClass` representing the modeled class to use, this `IClass` reference may be obtained from the model at runtime using the type system service (see page 147). This kind of list is useful when you require a strongly typed list of objects, for example if you wish to create a list in code rather than using an OCL (see page 22) expression and then present that list to the user via a `ReferenceHandle`. In this case a grid is connected to `rhRoot` which has its `StaticValueTypeName` property set to `Collection(Person)`.

```

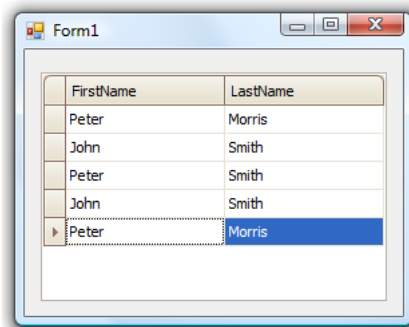
//Create three people
Person person1 = new Person(EcoSpace);
person1.FirstName = "Peter";
person1.LastName = "Morris";
Person person2 = new Person(EcoSpace);
person2.FirstName = "John";
person2.LastName = "Smith";
Person person3 = new Person(EcoSpace);
person3.FirstName = "Fred";
person3.LastName = "Jones";

```

```
//Create the object list
IVariableFactoryService vfs = EcoSpace.VariableFactory;
IObjectList objectsToPresent = vfs.CreateTypedObjectList(typeof(Person), true);

//Add only the locators to the list
objectsToPresent.Add(person1.AsIObject());
objectsToPresent.Add(person2.AsIObject());
objectsToPresent.Add(person3.AsIObject());
objectsToPresent.Add(person2.AsIObject());
objectsToPresent.Add(person1.AsIObject());

//Now present the list to the user interface
rhRoot.SetElement(objectsToPresent);
```



### Creating a variable list

A variable list is a collection of values cross referenced by name. Variable lists are useful when you need to execute an OCL expression which refers to multiple object instances. For example the following expression would select all bookings for a specific customer and flight combination, note that the prefix `var_` is purely optional and only used here for clarity

```
PropertyBooking.allInstances->select(customer = var_Customer)->select(flight = var_Flight)
```

The above expression could have been implemented with constants rather than variables, in fact when you create a variable list you may add constants or variables. However, variable lists may also be used in a similar way to how query parameters work in a SQL statement within a standard DB application.

```
IVariableFactoryService vfs = EcoSpace.VariableFactory;

//Create three people

Person person1 = new Person(EcoSpace);
person1.FirstName = "Peter";
person1.LastName = "Morris";
Person person2 = new Person(EcoSpace);
person2.FirstName = "John";
person2.LastName = "Smith";
Person person3 = new Person(EcoSpace);
person3.FirstName = "Fred";
person3.LastName = "Jones";
EcoSpace.UpdateDatabase();

//Create the variable
IModifiableVariableList variables = vfs.CreateVariableList();
IElement lastNameVar = vfs.CreateVariable(typeof(string));
```

```

variables.Add("var_LastName", lastNameVar);

//Define a list of values to loop through
string[] lastNames = new string[] {"Morris", "Smith", "Jones"};
//Define the OCL expression using var_LastName
string ocl = "Person.allInstances->select(lastName = var_LastName)";

//Now loop through each value
foreach (string currentLastName in lastNames)
{
    //Set the value of the variable
    lastNameVar.AsObject = currentLastName;
    //Execute the query
    IList<Person> people = EcoSpace.OclPs.Execute(null, variables, ocl, -1,
0).GetAsIList<Person>();
    //Now show the result
    foreach (Person currentPerson in people)
    {
        string message = string.Format("{0} -> {1} {2}",
            currentLastName,
            currentPerson.FirstName,
            currentPerson.LastName);
        MessageBox.Show(message);
    }
}

```

The above example creates a variable of type System.String, the return value of CreateVariable is an IElement. The variable is then added to a variable list with the name `var_LastName`. Within the loop it is possible to set the value of the variable by setting `lastNameVar.AsObject`. When the OCL expression is evaluated the text `var_LastName` will substituted with the variable's value instead. Just like query parameters this approach prevents the developer from having to escape special query characters in order to ensure the query always remains valid (see SQL Injection - [http://en.wikipedia.org/wiki/SQL\\_injection](http://en.wikipedia.org/wiki/SQL_injection), see also <http://xkcd.com/327>).

The important parameters being passed to `OclPs.Execute` are `variables` and `ocl`. The other parameters will be explained in the IOclPsService (see page 25) section.

## 2.7 Query services

These ECO services process queries that are based on the Object Constraint Language (OCL) in order to retrieve information from the ECO cache. When an EcoSpace is persistent ECO will also translate the OCL query into the query language of the data storage (usually SQL) to ensure that the data necessary to process the query is loaded into the ECO cache.

As not all OCL operations are supported by all target data storages OCL query support is cleanly separated into three distinct services. Each service is described in a particular order; the service being described is capable of processing a superset of any previously described OCL services.

### Format of OCL expressions

All OCL expressions start with a root context. This context may be a modeled class type, an instance of a modeled class, a single element (such as an integer or string), or a collection. The following expression shows how to use the modeled Person class to start the expression, and return all instances of that class:

```
Person.allInstances
```

The above expression would result in a collection of Person instances. From this context it is possible to either perform a collection operation or a member operation. Like many OOP languages to identify a member the context and member name are separated by a period, collection operations are separated by the token ->

```
//Returns the first person in the collection
Person.allInstances->first

//Returns the DateOfBirth of the first person in the collection
Person.allInstances->first.dateOfBirth
```

When a member name is appended directly to a collection the OCL parser will automatically iterate through each element within the collection and return a collection of the member specified.

```
//Returns a collection of DateTime, one for each person in the collection
Person.allInstances.dateOfBirth
```

### OCL expressions with a specific context

It is possible to evaluate OCL expressions on an individual instance of a modeled class. For example when defining OCL based validity constraints on a class those expressions would be evaluated in the context of an instance of that class. When evaluating against an instance the context is assumed to be that instance. The keyword **self** may be used within the expression to identify the root object

```
self.dateOfBirth
```

The self keyword is case sensitive and must always be written in lowercase. When evaluating an expression against a root this does not mean that the OCL expression must start with that root object. At any point within an OCL expression it is possible to reference a modeled class by its name and use that as the context of the expression.

```
//using self other than as the context of the expression
//self references an instance of Customer
Booking.allInstances->select(bookedBy = self)

//Returns the same result as
self.bookings
```

The above two expressions will return the same result. The first expression is less efficient than the second as it will first retrieve all instances of Booking and then filter the list down to only those booked by the root object (Customer), the second expression uses the Customer as the context of the expression to find its bookings which results in retrieving only bookings assigned to this Customer and not having to filter the list afterwards.

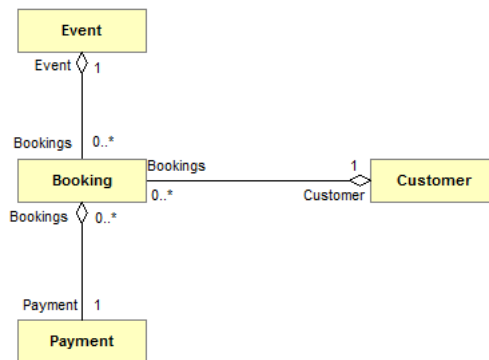
### Changing context within an expression

In the previous examples you can see that you can switch the context to a class despite having a root object to evaluate against. It is in fact possible to switch the context to a class at any point within an expression

```
self.eventsAttended->intersection(
    Event.allInstances->orderDescending(date)->subSequence(1, 5)
)
```

This more complicated expression uses a Customer as the root to evaluate the expression against (and therefore by default the initial context). It switches the context to all events attended by that customer and then filters that event list down to include only the last 5 events held. The last 5 events held are determined by selecting all Event instances, ordering the collection by their date (descending), and then selecting the top 5.

### Members with the same name as a class



In this simple model it makes sense that the association from a booking to the Payment class should be named "Payment", the association from a booking to the event should be named "Event", and that the association from Booking to Customer should be named "Customer".

When evaluating an expression such as the following (using Customer as the root)

```
bookings->orderBy(date)
```

it is quite clear that the start of the expression refers to the member Customer.Bookings. When evaluating an expression that starts with a member that has the same name as a class in your model it is not clear whether you intend the OCL evaluator to interpret the token as the member name or if you intend to switch the context to another class.

```
//Error - Event is interpreted as a class
event.cancelled
```

In such a case you would precede the member name explicitly with the keyword self.

```
self.event.cancelled
```

By now you may have noticed that all of the examples in this section use a lowercase initial letter for members and an uppercase initial letter for class names. This is a standard in the OCL and although it is not enforced in ECO it does help to clarify your intentions. It is good practise to precede association names with the self keyword when the association name is singular (event) rather than plural (events).

### Aliases

Building on the following example of explicitly identifying a token as a member and not a class there is a situation where it is not logical to precede the member name with the self keyword. The following OCL expression returns a collection of events a Customer has bookings for.

```
//Using a customer instance as the root
self.bookings.event
```

In this expression it is clear that the token "event" refers to the association Booking.Event because it is connected directly with a period, identifying it as a member; the previous expression would return a collection of Event instances. However, the following expression (using a Booking as the root) is not valid and will not evaluate at all.

```
self.bookings->select(event.cancelled)
```

This expression will not evaluate because the token "event" is not preceded with a period, identifying it as a member of Booking, but unlike in the previous scenario it is not possible to precede "event" with the self keyword because self will always refer to the root that the expression is being evaluated against; in this case that would be an instance of Customer, and Customer does not have an Event association.

The solution to this is to use an alias for each booking, so that the "event" token can be replaced with "alias.event" which

then clearly identifies it as a member and not a class. Aliases are identified with the | (pipe) symbol and look like this

```
//Long format
self.bookings->select(currentBooking | currentBooking.event.cancelled)

//Shorter format
self.bookings->select(b | b.event.cancelled)
```

## Comparisons

Comparisons in the OCL are made using a single equals sign.

```
self.bookings->select(date = DateTime.today)
```

## 2.7.1 IOclPsService

OclPs is an abbreviation for Object Constraint Language - (evaluated in) Persistent Storage. The IOclPsService translates OCL expressions into the data storage specific query language (usually SQL) in order to perform queries without first having to load object contents into the local ECO cache. The IOclPsService never retrieves any object data, instead it always returns a result type of `IObjectList`, which is a list of "object locators", an object's contents are not loaded until you reference the .NET object from the locator, for example

```
IObject myLocator = { retrieved from somewhere else };
//ECO will ensure the object is loaded if not already
Console.WriteLine(myLocator.GetValue<Person>().FirstName);
```

This is an important detail because it means that the only information returned from the data storage is a list of object ID's rather than the entire contents of the objects. Without this ability evaluating OCL could prove to be very expensive. Take a simple harmless looking OCL query such as the following

```
Person.allInstances->select(dateOfBirth = #1978-01-01)
```

If this query were to be evaluated in memory ECO would first have to load every instance of the `Person` class into memory and then filter the list down to only those with the specific date of birth. If there were millions of `Person` objects in the data storage this would obviously require a lot of memory and also a lot of network bandwidth.

1. [DB] Select FirstName, LastName, etc from Person;
2. [ECO] Receive row data for millions of Person objects
3. [ECO] Filter the list where the DateOfBirth matches
4. [App] Receive a collection of object locators that match the criteria

When this same query is executed with the IOclPsService the filtering will be performed by the data storage and only the ID's of the matching objects will be returned.

1. [DB] Select FirstName, LastName, etc from Person WHERE DateOfBirth = {format specific to the database};
2. [ECO] Receive only the IDs of only the rows that match the criteria
3. [App] Receive a collection of object locators that match the criteria

Note: The ECO component "OclPsHandle" uses the IOclPsService to execute its expression.

As you can see there is a big gain from using the IOclPsService when there is a large number of objects involved such as *SomeClass.allInstances* where it is anticipated there will be a lot of object instances (*Country.allInstances* would be okay, but *Person.allInstances* might prove problematic). This also includes "rooted" queries such as *self.Residents* where "self" is in instance of Country, in this case it is anticipated that a country is likely to have many residents so you would really want any `->select(...)` to be performed in the data storage rather than in memory.

### Restrictions of the IOclPsService

1. It does not take into account unsaved changes.	It is important to note that as the filtering is performed by the data storage it is only capable of providing a list of objects that have been persisted. If for example you were to modify the DateOfBirth of an already persisted Person in your local EcoSpace cache so that it does not match the date #1978-01-01, and then not persist those changes, when you execute the OclPs expression <i>Person.allInstances-&gt;select(dateOfBirth = #1978-01-01)</i> the result would still include the modified Person. There is a way of ensuring the collection represents both changes that have and have not yet been persisted using the in-memory OCL expression <i>AllLoadedObjects</i> , which will be covered in the section IOclService (see page 44).
2. It can only return a collection of object locators.	The result type is always an IObjectList. This means that it is not possible to end your OCL query with operators such as <code>-&gt;size</code> to return the number of items in the list, nor can you for example return a list of unique values using <code>-&gt;collect(dateOfBirth)</code> .
3. It supports only a subset of OCL.	It is not possible to map all OCL operations to SQL for example. As a consequence not all OCL operations are supported.
4. Derived members are not supported.	Members in the model marked as derived are calculated in-memory upon request. These values are never persisted to the data storage and as a consequence it is not possible to use them in OCL expressions evaluated by this service.

### Executing queries

This service has a number of overloaded methods for evaluating OCL expressions, each of which is named *Execute()*. The most simple overload accepts only an OCL expression to evaluate.

```
IObjectList locators;
string query = "Person.allInstances->select(dateOfBirth = #1978-01-01)";
locators = EcoSpace.OclPs.Execute(query);
```

The above example selects the object locators only for the Person objects that were born on January the 1st, 1978. Note that no Person data is returned from this query. Any time you access *locator.AsObject* or *locator.GetValue<T>* the individual object will be loaded. To retrieve multiple objects in as few DB trips as possible you can either use the *GetAsIList<T>()* method on the result or use the *EnsureRange()* method on the IPersistenceService (see page 133) to pre-load. Note that evaluating an in-memory query against the result using the IOclService will also fetch object data into the local EcoSpace cache. The following example not only identifies which Person objects match the criteria but also loads the objects' contents into the cache.

```
IList<Person> people;
string query = "Person.allInstances->select(dateOfBirth = #1978-01-01)";
people = EcoSpace.OclPs.Execute(query).GetAsIList<Person>();
```

The next overload is similar to the first except that it additionally takes two integer parameters.

Name	Type	Purpose
maxAnswers	Int32	Limits the number of object locators returned.
offset	Int32	Zero based index of the first object in the list to return.

This overload is useful for either data paging (in which case you should have an `->orderBy(...)` operation in your OCL expression to ensure a consistent order) or simply to reduce the number of objects returned on a search screen for example.

```
int maxAnswers = 50;

IList<Person> people;
string query = "Person.allInstances->select(dateOfBirth = #1978-01-01)";
people = EcoSpace.OclPs.Execute(query, maxAnswers, 0).GetAsIList<Person>();
```

The next overload expects not only an OCL expression to evaluate but also a root context to evaluate it against. The root is an `IElement`, which means that the context could be either a single instance of a modeled business class or a collection of instances. The root provided is always referenced as the OCL keyword `self` in expressions. Note that although `IElement` can be used to represent other values such as strings, integers, and dates it is not possible to use these as a root context for evaluating an OclPs expression.

```
City city = { retrieved from somewhere else };
IList<Person> people;
string query = "self.residents->select(dateOfBirth = #1978-01-01)";
people = EcoSpace.OclPs.Execute(city.AsIObject(), query).GetAsIList<Person>();

/*
Equivalent to SQL
select FirstName, LastName, etc from Person
where City = 132232 and DateOfBirth = { DB specific date format};
*/
```

It is not mandatory to use the root context as the root of the OCL evaluation. The following example produces the same result as the previous example despite starting the OCL expression with `Person.allInstances`.

```
City city = { retrieved from somewhere else };
IList<Person> people;
string query = "Person.allInstances->select(city = self)->select(dateOfBirth = #1978-01-01)";
people = EcoSpace.OclPs.Execute(city.AsIObject(), query).GetAsIList<Person>();
```

The final overload accepts a number of additional parameters.

Name	Type	Purpose
root	IElement	Acts as a context for the keyword <code>self</code> in OCL expressions. This parameter may be null.
variableList	IExternalVariableList	This parameter provides a variable list which is used when parsing the OCL expression. Any unidentified part of the expression that looks like it should be a literal value is checked against this variable list to see if it is a named value. This parameter may be null.
expression	String	The OCL expression to evaluate.
maxAnswers	Int32	The total number of answers to return.



offset	Int32	The zero based index of the first result to return.
--------	-------	---

The `IVariableFactoryService` (see page 18) section of this document provides an example for the use of this overload.

## 2.7.1.1 OCL operations supported by IOclPsService

### 2.7.1.1.1 +

Source	Int32
Parameters	Int32 value
Result	Int32

#### Description

Returns the result of adding the specified value to the source.

#### Example

```
1 + 2
```

#### Notes

Additional overloads exist for Int64 and Double.

### 2.7.1.1.2 -

Source	Int32
Parameters	Int32 value
Result	Int32

#### Description

Returns the result of subtracting the specified value from the source.

#### Example

```
3 - 2
```

#### Notes

Additional overloads exist for Int64 and Double.

### 2.7.1.1.3 \*

<b>Source</b>	Int32
<b>Parameters</b>	Int32 factor
<b>Result</b>	Int32

#### Description

Returns the result of multiplying the source by the factor.

#### Example

```
5 * 5
```

#### Notes

Additional overloads exist for Int64 and Double.

### 2.7.1.1.4 /

<b>Source</b>	Int32
<b>Parameters</b>	Int32 divisor
<b>Result</b>	Int32

#### Description

Returns the result of dividing the source by the divisor.

#### Example

```
10 / 2
```

#### Notes

Additional overloads exist for Int64 and Double.

### 2.7.1.1.5 <

<b>Source</b>	<Any>
<b>Parameters</b>	<Any> value
<b>Result</b>	Boolean

**Description**

Returns true if the source is less than the parameter.

**Example**

```
6 < 12
```

**2.7.1.1.6 <=**

<b>Source</b>	<Any>
<b>Parameters</b>	<Any> value
<b>Result</b>	Boolean

**Description**

Returns true if the source is less than or equal to the parameter.

**Example**

```
6 <= 12
```

**2.7.1.1.7 <>**

<b>Source</b>	<Any>
<b>Parameters</b>	<Any> value
<b>Result</b>	Boolean

**Description**

Returns true if the source is not equal to the parameter.

**Example**

```
6 <> 12
```

**2.7.1.1.8 =**

<b>Source</b>	<Any>
<b>Parameters</b>	<Any> value
<b>Result</b>	Boolean

**Description**

Returns true if the source is equal to the parameter.

**Example**

```
1 = 1
```

**2.7.1.1.9 >**

<b>Source</b>	<Any>
<b>Parameters</b>	<Any> value
<b>Result</b>	Boolean

**Description**

Returns true if the source is greater than the parameter.

**Example**

```
12 > 6
```

**2.7.1.1.10 >=**

<b>Source</b>	<Any>
<b>Parameters</b>	<Any> value
<b>Result</b>	Boolean

**Description**

Returns true if the source is greater than or equal to the parameter.

**Example**

```
12 >= 6
```

**2.7.1.1.11 AllInstances**

<b>Source</b>	<Type>
<b>Parameters</b>	
<b>Result</b>	Collection(<Instance>)

**Description**

Returns all instances of the source type. The type can be a business class such as Person. In OCL and EAL the type can also be an enumeration type used in the model such as Gender or OrderState

**Example**

```
Person.allInstances
```

**2.7.1.1.12 Average**

<b>Source</b>	Collection(Decimal)
<b>Parameters</b>	
<b>Result</b>	Decimal

**Description**

Returns the average of a collection of numerical values.

**Example**

```
Person.allInstances.age->average
```

**Notes**

Additional overloads exist for Int32, Int64, and Double; each of which return Double and not Decimal.

**2.7.1.1.13 Difference**

<b>Source</b>	Collection(<Any>)
<b>Parameters</b>	Collection(<Any>) value
<b>Result</b>	Collection(<Any>)

**Description**

Returns a copy of the source collection minus all elements in the parameter.

**Example**

```
var_Person1.friends->difference(var_Person2.friends)
```

**Notes**

To remove a single element see Excluding.

**2.7.1.1.14 Div**

<b>Source</b>	Int32
<b>Parameters</b>	Int32 divisor
<b>Result</b>	Int32

**Description**

Returns the result of dividing the source by the divisor.

**Example**

```
10 div 2
```

**2.7.1.1.15 Exists**

<b>Source</b>	Collection(<Any>)
<b>Parameters</b>	Boolean expression
<b>Result</b>	Boolean

**Description**

Returns true if the parameter results in true for any of the items in the collection.

**Example**

```
self.friends->exists(f | f.Gender = Gender::Male)
```

**Notes**

The syntax within the brackets is similar to a lambda expression, where each item in the Collection(Person) is assigned the alias "f" for the rest of the expression.

**2.7.1.1.16 ForAll**

<b>Source</b>	Collection(<Any>)
<b>Parameters</b>	Boolean expression
<b>Result</b>	Boolean

**Description**

Returns true if the parameter results in true for every item in the collection.

**Example**

```
self.friends->forAll(f | f.Gender = Gender::Male)
```

The person in question only has male friends.

### 2.7.1.1.17 Implies

<b>Source</b>	Boolean
<b>Parameters</b>	Boolean comparison
<b>Result</b>	Boolean

#### Description

The following table shows possible permutations and the result.

Source	Parameter	Result
False	False	True
False	True	True
True	False	False
True	True	True

#### Example

If a Weather class has two boolean attributes "Raining" and "Cloudy" the following constraint would ensure that it must be cloudy in order for it to be raining.

```
raining implies cloudy
```

### 2.7.1.1.18 Includes

<b>Source</b>	Collection(<Any>)
<b>Parameters</b>	<Any> value
<b>Result</b>	Boolean

#### Description

Returns true if the parameter exist in the source.

#### Example

```
self.friends->includes(self.Father)
```

The person in question is friends with his/her father.

### 2.7.1.1.19 Intersection

<b>Source</b>	Collection(<Any>)
<b>Parameters</b>	Collection(<Any>) value
<b>Result</b>	Collection(<Any>)

**Description**

Returns a collection of all elements in the source that are also in the parameter.

**Example**

```
self.debtors->intersection(self.creditors)
```

Returns a collection of people who are both debtors and creditors.

**2.7.1.1.20 IsEmpty**

<b>Source</b>	Collection(<Any>)
<b>Parameters</b>	
<b>Result</b>	Boolean

**Description**

Returns true if a collection has no elements in it.

**Example**

```
self.debtors->isEmpty
```

Returns True if the person has no debtors.

**Notes**

IsEmpty may also be used on an association to a single object to check if it has a value. It is recommended that IsEmpty is used in such a case rather than comparing the associated object with nil.

```
self.linkToSingleObject = nil [Incorrect]
self.linkToSingleObject->isEmpty [Correct]
```

**2.7.1.1.21 IsNull**

<b>Source</b>	<Any>
<b>Parameters</b>	
<b>Result</b>	Boolean

**Description**

Returns true if the source is null.

**Example**

```
self.firstName.isNull
```



**Notes**

To check an association to a single object use isEmpty.

**2.7.1.1.22 Length**

<b>Source</b>	String
<b>Parameters</b>	
<b>Result</b>	Int32

**Description**

Returns the length of a string.

**Example**

```
Person.allInstances->select(firstName.length < 2)
```

**2.7.1.1.23 MaxValue**

<b>Source</b>	Collection(Int32)
<b>Parameters</b>	
<b>Result</b>	Int32

**Description**

Returns the largest value found in the source collection.

**Example**

```
Person.allInstances->select(height = Person.allInstances.height->maxValue)
```

Selects all of the tallest people.

**Notes**

Additional overloads exist for Int64, Double, and Decimal.

**2.7.1.1.24 MinValue**

<b>Source</b>	Collection(Int32)
<b>Parameters</b>	
<b>Result</b>	Int32

**Description**

Returns the minimum value found in the source.

**Example**

```
Person.allInstances->select(height = Person.allInstances.height->minValue)
```

Selects all of the shortest people.

**Notes**

Additional overloads exist for Int64, Double, and Decimal.

**2.7.1.1.25 Mod**

<b>Source</b>	Int32
<b>Parameters</b>	Int32 modulo
<b>Result</b>	Int32

**Description**

Returns the result of performing a modulus on the source using the modulo.

**Example**

```
17 mod 10
```

Returns 7.

**2.7.1.1.26 Not**

<b>Source</b>	Boolean
<b>Parameters</b>	
<b>Result</b>	Boolean

**Description**

Returns the result of performing a logical NOT operation on the source.

**Example**

```
User.allInstances->select(not suspended)
```

### 2.7.1.1.27 NotEmpty

<b>Source</b>	Collection(<Any>)
<b>Parameters</b>	
<b>Result</b>	Boolean

#### Description

Returns true if a collection has at least one element in it.

#### Example

```
self.debtors->notEmpty
```

Returns True if the person has debtors.

#### Notes

NotEmpty may also be used on an association to a single object to check if it has a value. It is recommended that NotEmpty is used in such a case rather than comparing the associated object with nil.

```
self.linkToSingleObject <> nil [Incorrect]
self.linkToSingleObject->notEmpty [Correct]
```

### 2.7.1.1.28 OrderBy

<b>Source</b>	Collection(<Any>)
<b>Parameters</b>	<Any> expression
<b>Result</b>	Collection(<Any>)

#### Description

Returns a collection based on the source which has been ordered into ascending order based on the value in the parameter.

#### Example

```
self.debtors->orderBy(oldestUnpaidDebt)
```

Returns all debtors in order of the debtor with the oldest unpaid debt first.

```
self.debtors->orderBy(lastName, firstName)
```

Returns all debtors ordered by their last name, for debtors with the same last name, the first name is used as a secondary key

#### Notes

Using OrderBy it is possible to order by multiple keys, but only in ascending order. To order by multiple items in different directions, use OrderGeneric.

### 2.7.1.1.29 OrderDescending

<b>Source</b>	Collection(<Any>)
<b>Parameters</b>	<Any> expression
<b>Result</b>	Collection(<Any>)

#### Description

Returns a collection based on the source which has been ordered into descending order based on the value in the parameter.

#### Example

```
self.debtors->orderDescending(amountOwed)
```

Returns all debtors in order of the debtor with the greatest amount of debt first.

#### Notes

Using OrderDescending it is possible to order by multiple keys, but only in ascending order. To order by multiple items in different directions, use OrderGeneric.

### 2.7.1.1.30 OrderGeneric

<b>Source</b>	Collection(<Any>)
<b>Parameters</b>	<Any> expression SortDirection direction
<b>Result</b>	Collection(<Any>)

#### Description

Returns a collection based on the source which has been ordered as specified by the parameters. OrderGeneric takes a minimum of two parameters; the first identifies the sort expression, and the second identifies the sort direction (ascending or descending). These two parameters may be repeated multiple times in order to provide sorting by multiple items.

#### Example

```
self.debtors->orderGeneric(oldestUnpaidDebt, OclSortDirection::ascending, amountOwed,
OclSortDirection::descending)
```

Returns all debtors primarily in order of the debtor with the oldest unpaid debt first, and then secondarily by the debtor owning the highest amount of money.

OldestUnpaidDebt	AmountOwed
2008-01-01	1000.00
2008-01-01	800.00
2008-01-01	700.00

2008-02-01	1000.00
2008-02-01	550.00

### 2.7.1.1.31 Reject

<b>Source</b>	Collection(<Any>)
<b>Parameters</b>	Boolean expression
<b>Result</b>	Collection(<Any>)

#### Description

Returns a collection based on the source excluding any elements where the boolean expression evaluated to True.

#### Example

```
self.friends->reject(age < 18)
```

This is functionally equivalent to the following

```
self.friends->select(age >= 18)
```

### 2.7.1.1.32 Select

<b>Source</b>	Collection(<Any>)
<b>Parameters</b>	Boolean expression
<b>Result</b>	Collection(<Any>)

#### Description

Returns a collection based on the source including only elements where the boolean expression evaluated to True.

#### Example

```
User.allInstances->select(u | u.firstName = 'Peter')->select(u | u.lastName = 'Morris')
```

Returns all users with the name "Peter Morris".

this is equivalent to

```
User.allInstances->select(u | (u.firstName = 'Peter') and (u.lastName = 'Morris'))
```

### 2.7.1.1.33 Size

<b>Source</b>	Collection(<Any>)
<b>Parameters</b>	
<b>Result</b>	Int32

**Description**

Returns the size of a collection.

**Example**

```
Person.allInstances->select(p | p.friends->size >= 5)
```

Returns all people with 5 or more friends.

**2.7.1.1.34 SqlLike**

<b>Source</b>	String
<b>Parameters</b>	String pattern
<b>Result</b>	Boolean

**Description**

Returns True if the source matches the pattern specified in the parameter. This is similar to the LIKE statement in SQL.

**Example**

```
Person.allInstances->select(p | p.lastName.sqlLike('%orris'))
```

Returns a collection of people who have a last name ending with "orris".

**Notes**

When evaluated in memory, this operation is case sensitive, but some databases treats the LIKE operator as a caseinsensitive operator. For guaranteed case insensitivity, use `sqlLikeCaseInsensitive`

% is used to denote a match with zero or more unknown letters

\_ is used to denote a match with exactly one unknown letter.

Value	Match
Morris	True
Norris	True
Orris	False

**2.7.1.1.35 SqlLikeCaseInsensitive**

<b>Source</b>	String
<b>Parameters</b>	String pattern
<b>Result</b>	Boolean

**Description**

Returns True if the source matches the pattern specified in the parameter. This is similar to the LIKE statement in SQL but case insensitive.

**Example**

```
Person.allInstances->select(lastName.sqlLikeCaseInsensitive('%orris'))
```

Returns a collection of people who have a last name ending with "orris", regardless of case.

**Notes**

Unlike the SqlLike operation the comparison is case insensitive.

Value	Match
Morris	True
Norris	True
Orris	True

**2.7.1.1.36 Sum**

<b>Source</b>	Collection(Int32)
<b>Parameters</b>	
<b>Result</b>	Int32

**Description**

Returns the sum of the values in the collection

**Example**

```
Person.allInstances->select(debts.value->sum > 10000)
```

Selects all people who have a debt of at least 10,000.

**Notes**

Additional overloads exist for Int64, Double, and Decimal.

**2.7.1.1.37 ToLower (String)**

<b>Source</b>	String
<b>Parameters</b>	
<b>Result</b>	String

**Description**

Converts a string value to lower case.

**Example**

```
Person.allInstances->select(firstName.toLower = 'peter')
```

Returns people with the first name "Peter", "pETeR" or any other case-variation of "peter".

**Notes**

When selecting you may wish to use `SqlLikeCaseInsensitive` as it permits the use of wildcards. The `ToLower` operation is mostly used for presenting data in a user interface in a specific format.

**2.7.1.1.38 ToUpper(String)**

<b>Source</b>	String
<b>Parameters</b>	
<b>Result</b>	String

**Description**

Converts a string value to upper case.

**Example**

```
Person.allInstances->select(firstName.toLower = 'PETER')
```

Returns people with the first name "Peter".

**Notes**

When selecting you may wish to use `SqlLikeCaseInsensitive` as it permits the use of wildcards. The `ToUpper` operation is mostly used for presenting data in a user interface in a specific format.

**2.7.1.1.39 Union**

<b>Source</b>	Collection(<Any>)
<b>Parameters</b>	Collection(<Any>) value
<b>Result</b>	Collection(<Any>)

**Description**

Returns a collection of all elements in the source combined with all elements in the parameter.



**Example**

```
self.debtors->union(self.creditors)
```

Returns a collection of people who are either a debtors or creditors of a person.

---

## 2.7.2 IOclService

Ocl is an abbreviation for Object Constraint Language. The IOclService evaluates OCL expressions in memory. Objects are fetched from the data storage automatically if they are required in order to evaluate the expression passed, the following expression evaluated against a PurchaseOrder would automatically load all PurchaseOrderLine objects related to the order if they were not already loaded into the local EcoSpace cache.

```
self.lines.value->sum
```

Due to the fact that the IOclService evaluates in memory it is advisable that you do not evaluate expressions such as *Class.allInstances* unless you are certain that there will be an acceptable number of instances. For example evaluating *Country.allInstances* would be acceptable whereas evaluating an expression such as *StockMovement.allInstances* would most likely be unacceptable due to the fact that you cannot guarantee that over time there will not be millions of StockMovement objects.

It is important to note that even OCL expressions which return a small number of results may still load a large number of objects into the local EcoSpace cache.

```
StockMovement.allInstances->select(id = 123456789)
```

1. [DB] Select {names of columns} from StockMovement
2. [ECO] Receive row data for millions of StockMovement objects
3. [ECO] Store the data in the local EcoSpace cache
4. [ECO] Filter the list where the id matches
5. [App] Receive a collection of object locators that match the criteria, in this case a single object out of over 100 million objects

When such an expression is required it is advisable to use the IOclPService (see page 25) instead.

**Evaluating OCL expressions**

This service has a number of overloaded methods for evaluating OCL expressions, each of which is named *Evaluate()*. The most simple overload accepts only an OCL expression to evaluate.

```
EcoSpace.Ocl.Evaluate("1 + 2");
```

The next overload accepts an additional parameter identifying a context for the evaluation.

Name	Type	Purpose
root	IElement	Identifies an element to be used as the context for the evaluation and also to determine the value to use wherever the evaluator encounters the keyword <i>self</i> in the expression.  For in-memory evaluations the context may be any kind of IElement <ul style="list-style-type: none"> <li>• Value types such as Int32, String, etc</li> <li>• Instances of modeled classes (as per the following example)</li> <li>• A collection of IElement such as a list of instances or a list of values</li> </ul>

```
Person person1 = new Person(EcoSpace);
person1.DateOfBirth = DateTime.Today;
EcoSpace.Ocl.Evaluate(person1.AsIObject(), "age + 1")
```

The next overload accepts provides a way of identifying multiple values as parameters

Name	Type	Purpose
root	IElement	Identifies an element to be used as the context for the evaluation.
expression	String	The OCL expression to evaluate.
variableList	IExternalVariableList	A collection of variables to use when parsing the OCL expression.

```
Person person1 = new Person(EcoSpace);
person1.DateOfBirth = DateTime.Today;

IModifiableVariableList vars = EcoSpace.VariableFactory.CreateVariableList();
vars.AddConstant("var_Years", 2);

EcoSpace.Ocl.Evaluate(person1.AsIObject(), "age + var_Years", vars);
```

Note: There is an additional overload which is the same as this except it does not have the initial *root* parameter.

## Evaluating and subscribing

The IOclService has a number of overloaded methods named *EvaluateAndSubscribe()*. These methods are essentially the same as the *Evaluate* methods except that they additionally take two parameters of the type *ISubscriber*. The subscription mechanism is covered in more detail in the subscriptions (see page 144) section of this document so will not be covered in great depth here. The only method that involves subscriptions which will be covered here is the *GetDerivedElement()* method.

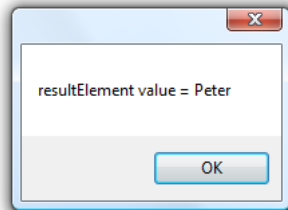
As ECO parses an OCL expression to evaluate it it accesses various element values held within the local cache, elements such as object instances, attributes, and roles are known as domain elements because they are physical parts of the modeled domain (Person, Person.FirstName, Person.LastName, etc). When an OCL expression returns a domain element you can be sure that whenever the value of the element changes your result will also change, for example:

```
Person person1 = new Person(EcoSpace);
person1.FirstName = "Fred";

string ocl = "firstName";
IElement resultElement = EcoSpace.Ocl.Evaluate(person1.AsIObject(), ocl);
```

```
person1.FirstName = "Peter";
MessageBox.Show("resultElement value = " + resultElement.GetValue<string>());
```

At the point the evaluation is performed the person's first name is "Fred", the element returned by the evaluation is stored in the local variable *resultElement*. The person's first name is then changed to "Peter" and a message box is then used to show the value held by the element returned earlier.

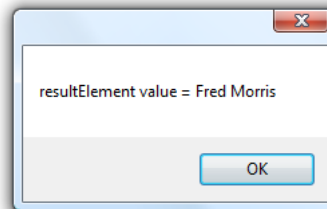


Here you see that the message box shows the new value "Peter" and not the initial value "Fred". This is because the result of the evaluation was a domain element, meaning that the expression identified a member of a class rather than merely a calculated value. To clarify this take a look at the following example

```
Person person1 = new Person(EcoSpace);
person1.FirstName = "Fred";
person1.LastName = "Morris";

string ocl = "firstName + ' ' + lastName";
IElement resultElement = EcoSpace.Ocl.Evaluate(person1.AsIObject(), ocl);

person1.FirstName = "Peter";
MessageBox.Show("resultElement value = " + resultElement.GetValue<string>());
```



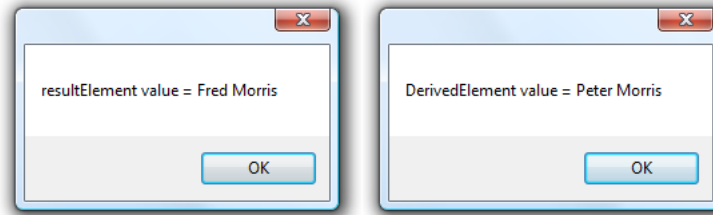
Here you see that the message box now shows the result as it would have been at the time of evaluation rather than reflecting any changes made since. This is because the result is evaluated as the values of two domain elements combined together with a single space between them, the result itself is not a domain element but a new element constructed solely for the purpose of holding the result. If you require the value of your element to change to reflect modifications after it was evaluated you can use the *GetDerivedElement()* method.

```
Person person1 = new Person(EcoSpace);
person1.FirstName = "Fred";
person1.LastName = "Morris";

string ocl = "firstName + ' ' + lastName";
IElement resultElement = EcoSpace.Ocl.Evaluate(person1.AsIObject(), ocl);
IElement derivedElement = EcoSpace.Ocl.GetDerivedElement(person1.AsIObject(), ocl);

person1.FirstName = "Peter";

MessageBox.Show("resultElement value = " + resultElement.GetValue<string>());
MessageBox.Show("DerivedElement value = " + derivedElement.GetValue<string>());
```



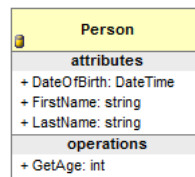
Here you can see that the derived element's value changes as the values it is derived from change. Subscriptions (see page 144) are covered in another part of this document.

### Combining evaluations

It is sometimes necessary to evaluate expressions in memory due to one or more of the following reasons:

1. You need to select on a derived (calculated) member or the result of a method, and the IOclPsService does not support this functionality.
2. You need to include new and/or modified object instances in the local EcoSpace cache that also meet this criteria.

The following OCL example illustrates this point.



*GetAge()* is defined as a method with *IsQuery=True*. This is a useful trick for when you want to create a calculated member in code but not all of the elements involved in calculating its result are capable of notifying subscribers when they change - in this example there is no way to subscribe to *DateTime.Today* using an ECO *ISubscriber* so a method is created instead with *IsQuery* set to *True* so that the *IOclService* knows it may execute the query without causing any side affects.

```
public int GetAge()
{
    DateTime today = DateTime.Today;
    int result = today.Year - DateOfBirth.Year;
    if (today.DayOfYear < DateOfBirth.DayOfYear)
        result--;
    return result;
}
```

Now consider the following OCL expression:

```
Person.allInstances->select(firstName.sqlLikeCaseInsensitive('Pete%'))->select(GetAge()
>= 18)
```

It is not possible to evaluate this expression using the *IOclPsService* because part of the expression refers to the method *GetAge()*. It is also not recommended that this expression is evaluated using the *IOclService* as this would load all instances of the *Person* class into memory in order to evaluate the rest of the expression, and this could be very detrimental

to system performance if there are many object instances. The solution to this is to perform two expression evaluations, one against each of the two services.

```
string psCriteria =
    "Person.allInstances" +
    "->select(firstName.sqlLikeCaseInsensitive('Pete%'))";
IObjectList people = EcoSpace.OclPs.Execute(psCriteria);

string memoryCriteria = "self->select(GetAge() >= 18)";
people = EcoSpace.Ocl.Evaluate(people, memoryCriteria);

IList<Person> result = people.GetAsIList<Person>();
```

The above code example first executes part of the expression as SQL and loads into memory only a collection of object locators, one for each object in the data storage that matched the criteria `firstName.sqlLikeCaseInsensitive('Pete%')`. The list of object locators is then used as the context for an in-memory evaluation to filter the collection down to only people who are at least 18 years old at the time of evaluating the expression. The problem with this example is that only objects identified as a match by the database will be considered for inclusion by the IOclService, this is because we are providing a specific list of object locators and then starting the expression with the `self` keyword. If there are instances of the Person class in the local cache which have not yet been saved then the data base has no way of knowing about this instance and including its object locator in the initial result; likewise if a previously saved object has been modified and the changes not yet saved the final result will not include objects which should be there.

One useful side affect of using the IOclPsService to retrieve a list of object locators is that these objects are then considered to be "loaded" even though we only have an object locator (identity of the object) and not the actual data for those objects. This is useful because there is an OCL operation available in the IOclService named `AllLoadedObjects` which can be used to evaluate expressions only against objects that have already had their identity loaded into the cache.

```
//Execute the query to load object locators into memory
string psCriteria =
    "Person.allInstances" +
    "->select(firstName.sqlLikeCaseInsensitive('Pete%'))";
EcoSpace.OclPs.Execute(psCriteria);

//Now evaluate only on objects already loaded
string memoryCriteria =
    "Person.allLoadedObjects" +
    "->select(firstName.sqlLikeCaseInsensitive('Pete%'))" +
    "->select(GetAge() >= 18)";
IElement people = EcoSpace.Ocl.Evaluate(memoryCriteria);

IList<Person> result = people.GetAsIList<Person>();
```

This is exactly the kind of approach you will see on the EcoDataSource component when creating ECO powered web applications, where it is possible to specify both a PsExpression property for evaluating the query as SQL on the database and also an Expression property which is evaluated against the result of the PsExpression.

### 2.7.2.1 OCL operations supported by IOclService

The IOclService supports all of the IOclPsService operations (see page 28) plus the following additional operations.

**2.7.2.1.1 -**

<b>Source</b>	DateTime
<b>Parameters</b>	TimeSpan value
<b>Result</b>	DateTime

**Description**

Returns the source DateTime with the specified TimeSpan parameter subtracted from it.

**Example**

```
self.dateOfBirth - #00:01
```

<b>Source</b>	DateTime
<b>Parameters</b>	DateTime
<b>Result</b>	TimeSpan

**Description**

Subtracts the DateTime parameter from the source and returns a TimeSpan.

**Example**

```
dateReturned - dateHired
```

<b>Source</b>	TimeSpan
<b>Parameters</b>	TimeSpan
<b>Result</b>	TimeSpan

**Description**

Subtracts the parameter from the source and returns a TimeSpan.

**Example**

```
endTime - startTime
```

### 2.7.2.1.2 +

<b>Source</b>	DateTime
<b>Parameters</b>	TimeSpan value
<b>Result</b>	DateTime

#### Description

Returns the source DateTime with the specified TimeSpan parameter added to it.

#### Example

```
self.dateOfBirth + #08:30
```

<b>Source</b>	TimeSpan
<b>Parameters</b>	TimeSpan
<b>Result</b>	TimeSpan

#### Description

Returns the source TimeSpan with the specified TimeSpan parameter added to it.

#### Example

```
endTime > (startTime + #01:00)
```

### 2.7.2.1.3 AllInstancesAtTime

<b>Source</b>	<Type>
<b>Parameters</b>	Int32 versionNumber
<b>Result</b>	Collection(<Instance>)

#### Description

Returns all instances of the source type for the given version number. See the IVersionService (see page 141) for more details.

#### Example

```
Person.allInstancesAtTime(0)
```

Returns a historical view of all Person instances that existed after the first ever call to UpdateDatabase.

### 2.7.2.1.4 AllLoadedObjects

<b>Source</b>	<Type>
<b>Parameters</b>	
<b>Result</b>	Collection(<Instance>)

#### Description

Returns all instances of the source type that have already been loaded into the local cache. This list will also include objects that have had their unique ID (object locator) loaded but not yet had their data contents loaded. No additional object locators will be retrieved from the data storage, however, accessing a member value via code or OCL will ensure that the instance's member data is in the cache.

#### Example

```
Person.allLoadedObjects
```

Returns all instances of Person that have already been loaded.

### 2.7.2.1.5 AllSubClasses

<b>Source</b>	<Type>
<b>Parameters</b>	
<b>Result</b>	Collection(String)

#### Description

Returns the names of all classes descended from the source type.

#### Example

```
Person.allSubClasses
```

### 2.7.2.1.6 AllSuperClasses

<b>Source</b>	<Type>
<b>Parameters</b>	
<b>Result</b>	Collection(String)

#### Description

Returns the names of all classes the source type descends from.



**Example**

```
Person.allSuperClasses
```

**2.7.2.1.7 AssociationEnds**

<b>Source</b>	<Type>
<b>Parameters</b>	
<b>Result</b>	Collection(String)

**Description**

Returns the names of all properties of the class that are association ends to another modeled class. This list includes both navigable and non-navigable association ends, and also includes association ends inherited from all super classes.

**Example**

```
Person.associationEnds
```

**2.7.2.1.8 AsString**

<b>Source</b>	<Any>
<b>Parameters</b>	
<b>Result</b>	String

**Description**

Represents the source as a string. When the source is an instance of a class it will evaluate the Default String Representation expression of the class.

**Example**

```
self.dateOfBirth.asString
```

**2.7.2.1.9 At**

<b>Source</b>	Collection(<Any>)
<b>Parameters</b>	Int32 index
<b>Result</b>	<Any>

**Description**

Returns the item within the collection at the specified index. The lower bound of the index is 1.

**Example**

```
Person.allInstances->at(1)
```

**Notes**

To specify a zero based index use ->at0

**2.7.2.1.10 AtTime**

<b>Source</b>	<Object>
<b>Parameters</b>	Int32 versionNumber
<b>Result</b>	<Object>

**Description**

Returns a historical view of the source object at the point in time identified by the version number. See the IVersionService (see page 141) for more details.

**Example**

```
self.atTime(0)
```

Returns a historical view of the source object that existed after the first ever call to UpdateDatabase.

**2.7.2.1.11 Attributes**

<b>Source</b>	<Type>
<b>Parameters</b>	
<b>Result</b>	Collection(String)

**Description**

Returns a collection of strings, each one being the name of a modeled property (UML attribute) on the specified class type. The list will contain attributes modeled on the current class plus any inherited from its base class; the list will not include any association ends.

**Example**

```
IElement attributes;
attributes = EcoSpace.Ocl.Evaluate("Class_1.attributes");
foreach (string current in attributes.GetAsIList<string>())
    MessageBox.Show(current);
```

**2.7.2.1.12 Collection operations**

### 2.7.2.1.12.1 Append

<b>Source</b>	Collection(<Any>)
<b>Parameters</b>	<Any> value
<b>Result</b>	Collection(<Any>)

#### Description

Returns the source collection with the parameter added to it.

#### Example

```
var_customer1.orders->append(var_customer2.orders)
```

Returns a collection of orders from customer1 with a single order from customer2 appended to the end of the list.

### 2.7.2.1.12.2 AsBag

<b>Source</b>	Collection(<Any>)
<b>Parameters</b>	
<b>Result</b>	Collection(<Any>)

#### Description

Evaluates the source and returns a collection based upon it that is capable of holding duplicate entries. By default associations are Sets, meaning that they do not allow duplicate values. This operation does not alter the source collection, it merely changes the context.

#### Example

```
//Products will only appear once
var_order1.lines.product->including(var_order2.lines.product)

//Duplicate products may appear
var_order1.lines.product->asBag->including(var_order2.lines.product)
```

This operation is not very useful in ECO. It is implemented as a part of complying with the OCL specification. In ECO, all object lists have an explicit order order.

### 2.7.2.1.12.3 AsCommaList

<b>Source</b>	Collection(String)
<b>Parameters</b>	
<b>Result</b>	String

#### Description

Takes a collection of strings as a source and returns a single string containing each of the values in the collection separated by commas.

**Example**

```

City city = new City(EcoSpace);

Person person1 = new Person(EcoSpace);
person1.City = city;
person1.FirstName = "Peter";

Person person2 = new Person(EcoSpace);
person2.City = city;
person2.FirstName = "Fred";

string ocl = "self.people.firstName->asCommaList";
string result =
    EcoSpace.Ocl.Evaluate(city.AsIObject(), ocl).GetValue<string>();
MessageBox.Show(result);

```

Returns the value

*Peter, Fred*

**2.7.2.1.12.4 AsSeparatedList**

<b>Source</b>	Collection(String)
<b>Parameters</b>	String separator
<b>Result</b>	String

**Description**

Takes a collection of strings as a source and returns a single string containing each of the values in the collection separated by the specified separator.

**Example**

```

City city = new City(EcoSpace);

Person person1 = new Person(EcoSpace);
person1.City = city;
person1.FirstName = "Peter";

Person person2 = new Person(EcoSpace);
person2.City = city;
person2.FirstName = "Fred";

string ocl = "self.people.firstName->asSeparatedList('+')";
string result =
    EcoSpace.Ocl.Evaluate(city.AsIObject(), ocl).GetValue<string>();
MessageBox.Show(result);

```

Returns the value

*Peter+Fred*

### 2.7.2.1.12.5 AsSequence

<b>Source</b>	Collection(<Any>)
<b>Parameters</b>	
<b>Result</b>	Collection(<Any>)

#### Description

Evaluates the source and returns a copy of the that permits manual reordering. Reordering the resulting element does not affect the source.

### 2.7.2.1.12.6 AsSet

<b>Source</b>	Collection(<Any>)
<b>Parameters</b>	
<b>Result</b>	Collection(<Any>)

#### Description

Returns a collection based on the source with all duplicates removed.

#### Example

```
self.orders.lines.product->asSet
```

Returns a distinct list of products sold to a specific customer.

#### Example

```
Person person1 = new Person(EcoSpace);
person1.FirstName = "Peter";

Person person2 = new Person(EcoSpace);
person2.FirstName = "Fred";

IObjectList originalPeople =
    EcoSpace.VariableFactory.CreateTypedObjectList(typeof(Person), true);
originalPeople.Add(person1.AsIObject());
originalPeople.Add(person1.AsIObject());
originalPeople.Add(person2.AsIObject());
originalPeople.Add(person2.AsIObject());

string ocl = "self->asSet";
IElement result =
    EcoSpace.Ocl.Evaluate(originalPeople, ocl);
IList<Person> people = result.GetAsIList<Person>();
foreach (Person person in people)
    MessageBox.Show(person.FirstName);
```

Will show only two message boxes.

### 2.7.2.1.12.7 Collect

<b>Source</b>	Collection(<Any>)
<b>Parameters</b>	expression
<b>Result</b>	Collection(<Any>)

#### Description

A new collection is created, and for each element in the source the parameter is added.

#### Example

```
private void TestCollect()
{
    CreateOrder(1);
    CreateOrder(2);
    CreateOrder(3);

    IElement result;
    result = EcoSpace.Ocl.Evaluate("Order.allInstances->collect(o |
o.orderLines->size)");
    foreach (int value in result.GetAsIList<int>())
        MessageBox.Show(value.ToString());
}

private void CreateOrder(int count)
{
    Order order = new Order(EcoSpace);
    for (int i = 0; i < count; i++)
    {
        OrderLine line = new OrderLine(EcoSpace);
        line.Order = order;
    }
}
```

If multiple arguments are specified in the collect-operation, the result will be a collection of tuples:

```
Person.allInstances->collect(firstName, lastName)
```

Will return a collection of tuples of pairs of strings

```
Person.allInstances->collect(firstName, lastName, friends->select(friends->size > 5))
```

Will return a collection of tuples that contains the firstname, lastname and a list of friends with more than 5 friends.

### 2.7.2.1.12.8 Count

<b>Source</b>	Collection(<Any>)
<b>Parameters</b>	<Any>
<b>Result</b>	Int32

#### Description

Returns the number of occurrences of the parameter in within the source collection.

```
self.friends.friends->count(self)
```

calculates how many of a persons friends who have the person listed as a friend.

### 2.7.2.1.12.9 Excluding

<b>Source</b>	Collection(<Any>)
<b>Parameters</b>	<Any> value
<b>Result</b>	Collection(<Any>)

#### Description

Returns a copy of the source minus the value specified in the parameter.

#### Example

```
Person.allInstances->excluding(self)
```

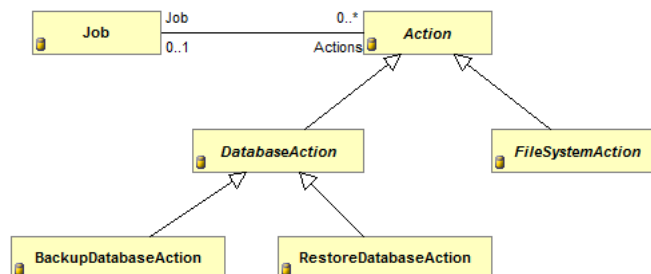
### 2.7.2.1.12.10 FilterOnType

<b>Source</b>	Collection(<Object>)
<b>Parameters</b>	Type requiredType
<b>Result</b>	Collection(<Object>)

#### Description

Returns a copy of the source; excluding any elements that are not of the type specified and not descended from the type specified.

#### Example



```
self.actions->filterOnType(DatabaseAction)
```

Returns all associated actions that are either DatabaseAction, BackupDatabaseAction, or RestoreDatabaseAction.

### 2.7.2.1.12.11 First

<b>Source</b>	Collection(<Any>)
<b>Parameters</b>	
<b>Result</b>	<Any>

**Description**

Returns the first element of the source collection.

**Example**

```
self.actions->first
```

**2.7.2.1.12.12 IncludesAll**

<b>Source</b>	Collection(<Any>)
<b>Parameters</b>	Collection(<Any>) value
<b>Result</b>	Boolean

**Description**

Returns true if every element in the parameter exists in the source.

**Example**

```
self.friends->includesAll(var_Person2.friends)
```

All of var\_Person2's friends are also the friends of the current person.

**2.7.2.1.12.13 Including**

<b>Source</b>	Collection(<Any>)
<b>Parameters</b>	<Any> value
<b>Result</b>	Collection(<Any>)

**Description**

Returns a copy of the source plus the value specified in the parameter.

**Example**

```
someContext.people->including(var_Person)
```

**2.7.2.1.12.14 IndexOf**

<b>Source</b>	Collection(<Any>)
<b>Parameters</b>	<Any> value
<b>Result</b>	Int32

**Description**

Returns the index of the parameter within the source collection. The first result in the collection is 1, if the parameter is not within the collection zero will be returned.



**Example**

```
self.purchaseOrder.lines->indexOf(self)
```

If PurchaseOrder.Lines is an ordered association then this OCL expression is a reliable way of determining the line number on the PurchaseOrderLine class.

**Notes**

To return a zero based index use ->indexOf0

**2.7.2.1.12.15 Last**

<b>Source</b>	Collection(<Any>)
<b>Parameters</b>	
<b>Result</b>	<Any>

**Description**

Returns the last element of the source collection.

**Example**

```
self.actions->last
```

**2.7.2.1.12.16 Prepend**

<b>Source</b>	Collection(<Any>)
<b>Parameters</b>	<Any> itemToAdd
<b>Result</b>	Collection(<Any>)

**Description**

Returns the source collection with the addition of the itemToAdd, which appears as the first element within the result.

**2.7.2.1.12.17 SubSequence**

<b>Source</b>	Collection(<Any>)
<b>Parameters</b>	<ul style="list-style-type: none"> <li>Int32 firstIndex</li> <li>Int32 numberOfItems</li> </ul>
<b>Result</b>	Collection(<Any>)

**Description**

Returns a subset of the source collection, starting at the first index specified and containing no more than the number of items specified. The first index available is 1.

### 2.7.2.1.12.18 SymmetricDifference

<b>Source</b>	Collection(<Any>)
<b>Parameters</b>	Collection(<Any>) comparison
<b>Result</b>	Collection(<Any>)

#### Description

Returns a collection of all items that appear in only the source or parameter, excluding any items that appear in both.

### 2.7.2.1.13 Compare

<b>Source</b>	Int32
<b>Parameters</b>	Int32 comparison
<b>Result</b>	Int32

#### Description

Compares the source to the parameter and returns either

-1	The source is less than the specified value.
0	The source and parameter are equal.
1	The source is greater than the specified value.

#### Example

```
age.Compare(18)
```

#### Notes

Overloads exist for Decimal, DateTime, TimeSpan, and String.

<b>Source</b>	String
<b>Parameters</b>	<ul style="list-style-type: none"> <li>String comparison</li> <li>Boolean ignoreCase</li> </ul>
<b>Result</b>	Int32

#### Description

Performs the same task except allows you to perform a case-insensitive comparison.

<b>Source</b>	String
<b>Parameters</b>	<ul style="list-style-type: none"> <li>• Int32 firstCharacterPositionOfSource</li> <li>• String comparison</li> <li>• Int32 firstCharacterPositionOfComparison</li> <li>• Int32 lengthOfSubstringOfComparison</li> </ul>
<b>Result</b>	Int32

**Description**

A substring of the source is used instead of the whole value using `ClrSubstring` (see page 102). It is then compared to a substring of the parameter.

<b>Source</b>	String
<b>Parameters</b>	<ul style="list-style-type: none"> <li>• Int32 firstCharacterPositionOfSource</li> <li>• String comparison</li> <li>• Int32 firstCharacterPositionOfComparison</li> <li>• Int32 lengthOfSubstringOfComparison</li> <li>• Boolean ignoreCase</li> </ul>
<b>Result</b>	Int32

**Description**

Identical to the previous overload except that the comparison may be made in a case insensitive manner.

**2.7.2.1.14 Constraints**

<b>Source</b>	<Object>
<b>Parameters</b>	
<b>Result</b>	Collection(Boolean)

**Description**

Evaluates each of the constraints on the specified business object. The return value itself isn't of much use, but it does provide a way of quickly determining the validity of an object's state.

**Example**

```
self.constraints->select(c | not c)->isEmpty
```

Returns True if the object has no broken constraints.

### 2.7.2.1.15 Create

This operation creates a value of a specific type. This enables your OCL to be specific about the type of value you are identifying in the expression.

#### Example

```
self.employees->select(salary >= Decimal.Create(52000.12))
```

In the OclPsService the Create operation may be used with the following types.

Type	Parameters
DateTime	---Overload 01---
	<ul style="list-style-type: none"> <li>• Int64 numberOfTicks</li> </ul>
	---Overload 02---
	<ul style="list-style-type: none"> <li>• Int32 year</li> <li>• Int32 month</li> <li>• Int32 day</li> </ul>
	---Overload 03---
	<ul style="list-style-type: none"> <li>• Int32 year</li> <li>• Int32 month</li> <li>• Int32 day</li> <li>• Int32 hour</li> <li>• Int32 minute</li> <li>• Int32 second</li> </ul>
	---Overload 04---
	<ul style="list-style-type: none"> <li>• Int32 year</li> <li>• Int32 month</li> <li>• Int32 day</li> <li>• Int32 hour</li> <li>• Int32 minute</li> <li>• Int32 millisecond</li> </ul>

Decimal	<p>---Overload 01---</p> <ul style="list-style-type: none"><li>• Single value</li></ul> <p>---Overload 02---</p> <ul style="list-style-type: none"><li>• Double value</li></ul> <p>---Overload 03---</p> <ul style="list-style-type: none"><li>• Int32 value</li></ul> <p>---Overload 04---</p> <ul style="list-style-type: none"><li>• UInt32 value</li></ul> <p>---Overload 05---</p> <ul style="list-style-type: none"><li>• Int64 value</li></ul> <p>---Overload 06---</p> <ul style="list-style-type: none"><li>• UInt64 value</li></ul> <p>---Overload 07---</p> <ul style="list-style-type: none"><li>• Int32 lo</li><li>• Int32 mid</li><li>• Int32 hi</li><li>• Boolean isNegative</li></ul>
---------	---

Guid	<p>---Overload 01---</p> <ul style="list-style-type: none"><li>• String value</li></ul> <p>---Overload 02---</p> <ul style="list-style-type: none"><li>• UInt32 a</li><li>• UInt16 b</li><li>• UInt16 c</li><li>• Byte d</li><li>• Byte e</li><li>• Byte f</li><li>• Byte g</li><li>• Byte h</li><li>• Byte i</li><li>• Byte j</li><li>• Byte k</li></ul> <p>---Overload 03---</p> <ul style="list-style-type: none"><li>• Int32 a</li><li>• Int16 b</li><li>• Int16 c</li><li>• Byte d</li><li>• Byte e</li><li>• Byte f</li><li>• Byte g</li><li>• Byte h</li><li>• Byte i</li><li>• Byte j</li><li>• Byte k</li></ul>
------	--

TimeSpan	---Overload 01--- <ul style="list-style-type: none"> <li>• Int64 numberOfTicks</li> </ul> ---Overload 02--- <ul style="list-style-type: none"> <li>• Int32 hours</li> <li>• Int32 minutes</li> <li>• Int32 seconds</li> </ul> ---Overload 03--- <ul style="list-style-type: none"> <li>• Int32 days</li> <li>• Int32 hours</li> <li>• Int32 minutes</li> <li>• Int32 seconds</li> </ul> ---Overload 04--- <ul style="list-style-type: none"> <li>• Int32 days</li> <li>• Int32 hours</li> <li>• Int32 minutes</li> <li>• Int32 seconds</li> <li>• Int32 milliseconds</li> </ul>
String	<ul style="list-style-type: none"> <li>• Char characterToRepeat</li> <li>• Int32 numberOfTimesToRepeat</li> </ul>

## 2.7.2.1.16 Date and time operations

### 2.7.2.1.16.1 AddDays

<b>Source</b>	DateTime
<b>Parameters</b>	Double days
<b>Result</b>	DateTime

#### Description

Returns the source with the specified number of days added to it.

#### Example

```
startTime.AddDays(7)
```

### 2.7.2.1.16.2 AddHours

<b>Source</b>	DateTime
<b>Parameters</b>	Double hours
<b>Result</b>	DateTime

**Description**

Returns the source with the specified number of hours added to it.

**Example**

```
startTime.AddHours(2.5)
```

**2.7.2.1.16.3 AddMilliseconds**

<b>Source</b>	DateTime
<b>Parameters</b>	Double milliseconds
<b>Result</b>	DateTime

**Description**

Returns the source with the specified number of milliseconds added to it.

**Example**

```
startTime.AddMilliseconds(100)
```

**2.7.2.1.16.4 AddMinutes**

<b>Source</b>	DateTime
<b>Parameters</b>	Double minutes
<b>Result</b>	DateTime

**Description**

Returns the source with the specified number of minutes added to it.

**Example**

```
startTime.AddMinutes(45)
```

**2.7.2.1.16.5 AddMonths**

<b>Source</b>	DateTime
<b>Parameters</b>	Int32 months
<b>Result</b>	DateTime

**Description**

Returns the source with the specified number of months added to it.



**Example**

```
startTime.AddMonths(1)
```

**2.7.2.1.16.6 AddSeconds**

<b>Source</b>	DateTime
<b>Parameters</b>	Double seconds
<b>Result</b>	DateTime

**Description**

Returns the source with the specified number of seconds added to it.

**Example**

```
startTime.AddSeconds(30)
```

**2.7.2.1.16.7 AddTicks**

<b>Source</b>	DateTime
<b>Parameters</b>	Int64 ticks
<b>Result</b>	DateTime

**Description**

Returns the source with the specified number of ticks added to it. The Int64 parameter is the number of 100-nanosecond ticks to add.

**Example**

```
startTime.AddTicks(1000)
```

**2.7.2.1.16.8 AddYears**

<b>Source</b>	DateTime
<b>Parameters</b>	Int32 years
<b>Result</b>	DateTime

**Description**

Returns the source with the specified number of years added to it.

**Example**

```
dateOfBirth.AddYears(100)
```

**2.7.2.1.16.9 Date**

<b>Source</b>	DateTime
<b>Parameters</b>	
<b>Result</b>	DateTime

**Description**

Returns only the date part of a specified DateTime value.

**Example**

```
self.dateOfBirth.date
```

**Notes**

It is possible to obtain the current date using the following OCL

```
DateTime.Now.Date
```

or

```
DateTime.Today
```

**2.7.2.1.16.10 Day**

<b>Source</b>	DateTime
<b>Parameters</b>	
<b>Result</b>	Int32

**Description**

Returns the day number of the date.

**Example**

```
#2025-12-21.day
```

Returns 21.

**2.7.2.1.16.11 Days**

<b>Source</b>	TimeSpan
<b>Parameters</b>	
<b>Result</b>	Int32

**Description**

Returns the number of whole days in the time span.

**Example**

```
TimeSpan.Create(99, 12, 00, 00).days
```

Returns 99.

**2.7.2.1.16.12 DayOfYear**

<b>Source</b>	DateTime
<b>Parameters</b>	
<b>Result</b>	Int32

**Description**

Returns the day number of the year.

**Example**

```
#2025-12-21.dayOfYear
```

Returns 355.

**2.7.2.1.16.13 DaysInMonth**

<b>Source</b>	DateTime
<b>Parameters</b>	<ul style="list-style-type: none"> <li>• Int32 year</li> <li>• Int32 month</li> </ul>
<b>Result</b>	Int32

**Description**

Returns the number of days in the specified month.

**Example**

```
DateTime.daysInMonth(2000, 2)
```

Returns 29.

### 2.7.2.1.16.14 Duration

<b>Source</b>	TimeSpan
<b>Parameters</b>	
<b>Result</b>	TimeSpan

#### Description

Returns the duration of the source time span. The result will always be a positive.

#### Example

```
TimeSpan.Create(-12, 00, 00).duration
```

Returns 12:00:00.

### 2.7.2.1.16.15 FormatDateTime

<b>Source</b>	DateTime
<b>Parameters</b>	String formatString
<b>Result</b>	String

#### Description

Returns the source data formatted using the specified format string.

#### Example

```
DateTime.now.formatDateTime('yyyy-MM-dd hh:mm:ss')
```

### 2.7.2.1.16.16 FromBinary

<b>Source</b>	DateTime
<b>Parameters</b>	Int64 serializedDateTimeValue
<b>Result</b>	DateTime

#### Description

Recreates a DateTime value from a serialized value.

#### Example

```
DateTime.FromBinary(DateTime.Now.ToBinary)
```

### 2.7.2.1.16.17 FromDays

<b>Source</b>	TimeSpan
<b>Parameters</b>	Double numberOfDays
<b>Result</b>	TimeSpan

#### Description

Creates a new TimeSpan from the number of days specified.

#### Example

```
TimeSpan.fromDays(99.5)
```

Returns 99:12:00:00.

### 2.7.2.1.16.18 FromFileTime

<b>Source</b>	DateTime
<b>Parameters</b>	Int64 fileDateTime
<b>Result</b>	DateTime

#### Description

Converts a file system date / time value to a DateTime.

#### Example

```
DateTime.fromFileTime(self.fileDateTime)
```

### 2.7.2.1.16.19 FromFileTimeUtc

<b>Source</b>	DateTime
<b>Parameters</b>	Int64 fileDateTime
<b>Result</b>	DateTime

#### Description

Converts the specified Windows file time to an equivalent UTC time.

#### Example

```
DateTime.fromFileTimeUtc(self.fileDateTime)
```

### 2.7.2.1.16.20 FromHours

<b>Source</b>	TimeSpan
<b>Parameters</b>	Double numberOfHours
<b>Result</b>	TimeSpan

#### Description

Creates a new TimeSpan from the number of hours specified.

#### Example

```
TimeSpan.fromHours(36)
```

Returns 1:12:00:00.

### 2.7.2.1.16.21 FromMilliseconds

<b>Source</b>	TimeSpan
<b>Parameters</b>	Double numberOfMilliseconds
<b>Result</b>	TimeSpan

#### Description

Creates a new TimeSpan from the number of milliseconds specified.

#### Example

```
TimeSpan.fromMilliseconds(1000)
```

Returns 00:00:01.

### 2.7.2.1.16.22 FromMinutes

<b>Source</b>	TimeSpan
<b>Parameters</b>	Double numberOfMinutes
<b>Result</b>	TimeSpan

#### Description

Creates a new TimeSpan from the number of minutes specified.

#### Example

```
TimeSpan.fromMinutes(1.5)
```

Returns 00:01:30.

### 2.7.2.1.16.23 FromSeconds

<b>Source</b>	TimeSpan
<b>Parameters</b>	Double numberOfSeconds
<b>Result</b>	TimeSpan

#### Description

Creates a new TimeSpan from the number of seconds specified.

#### Example

```
TimeSpan.FromSeconds(90)
```

Returns 00:01:30.

### 2.7.2.1.16.24 FromTicks

<b>Source</b>	TimeSpan
<b>Parameters</b>	Int64 ticks
<b>Result</b>	TimeSpan

#### Description

Creates a new TimeSpan from the number of ticks specified.

### 2.7.2.1.16.25 Hour

<b>Source</b>	DateTime
<b>Parameters</b>	
<b>Result</b>	Int32

#### Description

Returns the hour part of the source.

### 2.7.2.1.16.26 Hours

<b>Source</b>	TimeSpan
<b>Parameters</b>	
<b>Result</b>	Int32

**Description**

Returns the hour part of the source.

**2.7.2.1.16.27 InDateRange**

<b>Source</b>	DateTime
<b>Parameters</b>	<ul style="list-style-type: none"> <li>• DateTime firstDate</li> <li>• DateTime lastDate</li> </ul>
<b>Result</b>	Boolean

**Description**

Returns True if the source DateTime is  $\geq$  firstDate and  $\leq$  lastDate.

**Example**

```
self.appointments->select(a | a.dueDate.inDateRange(DateTime.today,
DateTime.today.addDays(1))
```

**2.7.2.1.16.28 InTimeRange**

<b>Source</b>	TimeSpan
<b>Parameters</b>	<ul style="list-style-type: none"> <li>• TimeSpan firstTime</li> <li>• TimeSpan lastTime</li> </ul>
<b>Result</b>	Boolean

**Description**

Returns True if the source TimeSpan is  $\geq$  firstTime and  $\leq$  lastTime.

**Example**

```
self.todaysAppointments->select(a | a.startTime.inTimeRange(DateTime.now,
DateTime.now.addHours(1))
```

**2.7.2.1.16.29 IsDaylightSavingTime**

<b>Source</b>	DateTime
<b>Parameters</b>	
<b>Result</b>	Boolean

**Description**

Returns a value indicating whether a specified date and time is within a daylight saving time period.

**Example**

```
DateTime.now.isDaylightSavingTime
```



**2.7.2.1.16.30 IsLeapYear**

<b>Source</b>	<Static operation> DateTime
<b>Parameters</b>	
<b>Result</b>	Boolean

**Description**

Returns a value indicating whether a specified year is a leap year.

**Example**

```
DateTime.IsLeapYear(2008)
```

**2.7.2.1.16.31 Millisecond**

<b>Source</b>	DateTime
<b>Parameters</b>	
<b>Result</b>	Int32

**Description**

Returns a number between 0 and 999 indicating the milliseconds part of the specified DateTime.

**2.7.2.1.16.32 Milliseconds**

<b>Source</b>	TimeSpan
<b>Parameters</b>	
<b>Result</b>	Int32

**Description**

Returns a number between 0 and 999 indicating the milliseconds part of the specified TimeSpan.

**2.7.2.1.16.33 Minute**

<b>Source</b>	DateTime
<b>Parameters</b>	
<b>Result</b>	Int32

**Description**

Returns a number between 0 and 59 indicating the minute part of the specified DateTime.

**2.7.2.1.16.34 Minutes**

<b>Source</b>	TimeSpan
<b>Parameters</b>	
<b>Result</b>	Int32

**Description**

Returns a number between 0 and 59 indicating the minute part of the specified TimeSpan.

**2.7.2.1.16.35 Month**

<b>Source</b>	DateTime
<b>Parameters</b>	
<b>Result</b>	Int32

**Description**

Returns a number between 1 and 12 indicating the month of the specified DateTime.

**2.7.2.1.16.36 Negate**

<b>Source</b>	TimeSpan
<b>Parameters</b>	
<b>Result</b>	TimeSpan

**Description**

Returns the negated value of the source. A positive TimeSpan will be result in a negative, and a negative in a positive.

**2.7.2.1.16.37 Now**

<b>Source</b>	<Static operation> DateTime
<b>Parameters</b>	
<b>Result</b>	DateTime

**Description**

Returns the current date and time.

**Example**

```
DateTime.now
```

**2.7.2.1.16.38 Second**

<b>Source</b>	DateTime
<b>Parameters</b>	
<b>Result</b>	Int32

**Description**

Returns a number between 0 and 59 indicating the second part of the specified DateTime.

**2.7.2.1.16.39 Seconds**

<b>Source</b>	TimeSpan
<b>Parameters</b>	
<b>Result</b>	Int32

**Description**

Returns a number between 0 and 59 indicating the second part of the specified TimeSpan.

**2.7.2.1.16.40 SumTime**

<b>Source</b>	Collection(TimeSpan)
<b>Parameters</b>	
<b>Result</b>	TimeSpan

**Description**

Produces a TimeSpan that is a sum of all entries in the collection.

**2.7.2.1.16.41 Ticks**

<b>Source</b>	DateTime
<b>Parameters</b>	
<b>Result</b>	Int64

**Description**

Returns the total number of ticks that represent the source value.

**Notes**

An overload also exists for TimeSpan.

**2.7.2.1.16.42 Time**

<b>Source</b>	DateTime
<b>Parameters</b>	
<b>Result</b>	TimeSpan

**Description**

Returns the time part of the source.

**2.7.2.1.16.43 TimeOfDay**

<b>Source</b>	DateTime
<b>Parameters</b>	
<b>Result</b>	TimeSpan

**Description**

Returns the time part of the source.

**2.7.2.1.16.44 TimeStampToTime**

<b>Source</b>	Int32
<b>Parameters</b>	
<b>Result</b>	DateTime

**Description**

This object-versioning operation will take an object time-stamp number (integer) and return the date and time at which the time stamp was created.

**Example**

```
self.objectTimeStamp.timeStampToTime
```

This example retrieves the current time stamp for the object instance (the last time it was modified) and returns the date and time of that time stamp.

**2.7.2.1.16.45 TimeToTimeStamp**

<b>Source</b>	DateTime
<b>Parameters</b>	
<b>Result</b>	Int32

**Description**

This object-versioning operation will take a date and time and return the appropriate time-stamp number that corresponds.

**Example**

```
DateTime.now.addDays(-1).timeToTimeStamp
```

### 2.7.2.1.16.46 ToBinary

<b>Source</b>	DateTime
<b>Parameters</b>	
<b>Result</b>	Int64

#### Description

Serializes the source to a 64 bit integer.

### 2.7.2.1.16.47 Today

<b>Source</b>	<Static operation> DateTime
<b>Parameters</b>	
<b>Result</b>	DateTime

#### Description

Returns only the date part of the current date.

### 2.7.2.1.16.48 ToFileTime

<b>Source</b>	DateTime
<b>Parameters</b>	
<b>Result</b>	Int64

#### Description

Returns the source converted to a Windows file system date/time.

### 2.7.2.1.16.49 ToFileTimeUtc

<b>Source</b>	DateTime
<b>Parameters</b>	
<b>Result</b>	Int64

#### Description

Returns the source converted to a Windows file system date/time.

### 2.7.2.1.16.50 ToLocalTime

<b>Source</b>	DateTime
<b>Parameters</b>	
<b>Result</b>	DateTime

#### Description

Returns the source converted to a local date/time.

**2.7.2.1.16.51 ToLongDateString**

<b>Source</b>	DateTime
<b>Parameters</b>	
<b>Result</b>	String

**Description**

Uses the Windows regional settings to convert the source into a long date string.

**2.7.2.1.16.52 ToLongTimeString**

<b>Source</b>	DateTime
<b>Parameters</b>	
<b>Result</b>	String

**Description**

Uses the Windows regional settings to convert the source into a long time string.

**2.7.2.1.16.53 ToShortDateString**

<b>Source</b>	DateTime
<b>Parameters</b>	
<b>Result</b>	String

**Description**

Uses the Windows regional settings to convert the source into a short date string.

**2.7.2.1.16.54 ToShortTimeString**

<b>Source</b>	DateTime
<b>Parameters</b>	
<b>Result</b>	String

**Description**

Uses the Windows regional settings to convert the source into a short time string.

**2.7.2.1.16.55 TotalDays**

<b>Source</b>	TimeSpan
<b>Parameters</b>	
<b>Result</b>	Double

**Description**

Returns the total number of days in the source, including the fractional part.

**2.7.2.1.16.56 TotalHours**

<b>Source</b>	TimeSpan
<b>Parameters</b>	
<b>Result</b>	Double

**Description**

Returns the total number of hours in the source, including the fractional part.

**2.7.2.1.16.57 TotalMilliseconds**

<b>Source</b>	TimeSpan
<b>Parameters</b>	
<b>Result</b>	Double

**Description**

Returns the total number of milliseconds in the source, including the fractional part.

**2.7.2.1.16.58 TotalMinutes**

<b>Source</b>	TimeSpan
<b>Parameters</b>	
<b>Result</b>	Double

**Description**

Returns the total number of minutes in the source, including the fractional part.

**2.7.2.1.16.59 TotalSeconds**

<b>Source</b>	TimeSpan
<b>Parameters</b>	
<b>Result</b>	Double

**Description**

Returns the total number of seconds in the source, including the fractional part.

**2.7.2.1.16.60 ToUniversalTime**

<b>Source</b>	DateTime
<b>Parameters</b>	
<b>Result</b>	DateTime

**Description**

Returns the source converted to a coordinated universal time.

### 2.7.2.1.16.61 UtcNow

<b>Source</b>	<Static operation> DateTime
<b>Parameters</b>	
<b>Result</b>	DateTime

#### Description

Returns the current date and time expressed as universal coordinated time.

#### Example

```
DateTime.utcnow
```

### 2.7.2.1.16.62 Year

<b>Source</b>	DateTime
<b>Parameters</b>	
<b>Result</b>	Int32

#### Description

Returns the whole number of years in the source.

### 2.7.2.1.17 EmptyList

<b>Source</b>	<Type>
<b>Parameters</b>	
<b>Result</b>	Collection(<Object>)

#### Description

Returns a strongly typed collection of the specified type, the result will have no elements in it.

#### Example

OCL for an association may look something like this

```
if (some condition) then
  self.orders
else
  Order.emptyList
endif
```

### 2.7.2.1.18 Existing

<b>Source</b>	<Instance>
<b>Parameters</b>	
<b>Result</b>	Boolean



**Description**

Returns true if the object still exists (and false if it has been deleted)

**2.7.2.1.19 ExternalId**

<b>Source</b>	<Instance>
<b>Parameters</b>	
<b>Result</b>	String

**Description**

Returns the ExternalId of the source. See the External ID Service (see page 4).

**2.7.2.1.20 If**

Provides a way of providing conditional evaluation.

**Example**

```
if (some condition)
  self.orders
else
  Order.emptyList
endif
```

**Notes**

Since every OCL expression needs to have a value, the "else-clause" of an if-statement is not optional as it is in most programming languages. The OCL evaluator will only evaluate the result of either the "then-clause" or the "else-clause" depending of the condition.

**2.7.2.1.21 Let**

The "Let" operation identifies the value of a variable in a statement.

The syntax for the let-statement is:

```
let <variablename> = <expression> in <expression>
```

The variable will be assigned the value of the first expression and can be referenced any number of times in the second expression

For example

```
let nameToFind = 'Peter' in Person.allInstances->select(firstName = nameToFind)
```

is the equivalent of

```
Person.allInstances->select(firstName = 'Peter')
```

This operator is useful for repeated use of a value.

```
let valueToUse = (some costly OCL evaluation) in
  Company.allInstances->select(
    (relevantMember > valueToUse) or (otherRelevantMember > valueToUse)
  )
```

If the evaluation of the value is costly then using the "let" operation prevents you from having to evaluate the value more than once.

## 2.7.2.1.22 Mathematical operations

### 2.7.2.1.22.1 Abs

Source	Int16
Parameters	
Result	Int16

#### Description

Returns the absolute value of a number.

#### Example

```
self.debt.abs
```

#### Notes

Additional overloads exist for SByte, Int16, Int16, Single, Double, and Decimal.

### 2.7.2.1.22.2 Acos

Source	Double
Parameters	
Result	Double

#### Description

Using the source as a Cosine value this operation will return its corresponding angle.

#### Example

```
self.cosineValue.acos
```

### 2.7.2.1.22.3 Asin

Source	Double
Parameters	
Result	Double

#### Description

Using the source as a Sine value this operation will return its corresponding angle.

**Example**

```
self.sineValue.asin
```

**2.7.2.1.22.4 Atan**

<b>Source</b>	Double
<b>Parameters</b>	
<b>Result</b>	Double

**Description**

Using the source as a tangent this operation will return its corresponding angle.

**Example**

```
self.tangent.atan
```

**2.7.2.1.22.5 Atan2**

<b>Source</b>	Double
<b>Parameters</b>	Double yCoOrdinate
<b>Result</b>	Double

**Description**

Uses the source as a relative X coordinate and the parameter as a relative Y coordinate. Using this coordinate the tangent is calculated and its corresponding angle is returned.

**Example**

```
self.xposition.atan2(yposition)
```

**2.7.2.1.22.6 BigMul**

<b>Source</b>	Int32
<b>Parameters</b>	Int32 factor
<b>Result</b>	Int64

**Description**

Multiplies the source by the factor and returns an Int64. This operation should be used when the source is an Int32 but the result is expected to be too large to be represented by an Int32.

**2.7.2.1.22.7 Ceiling**

<b>Source</b>	Double
<b>Parameters</b>	
<b>Result</b>	Double

**Description**

Rounds the source up to the closest whole number.

**Notes**

An overload also exists for Decimal.

**2.7.2.1.22.8 Cos**

<b>Source</b>	Double
<b>Parameters</b>	
<b>Result</b>	Double

**Description**

Using the source as an angle this operation will return its corresponding Cosine value.

**Example**

```
self.angle.cos
```

**2.7.2.1.22.9 Cosh**

<b>Source</b>	Double
<b>Parameters</b>	
<b>Result</b>	Double

**Description**

Using the source as an angle this operation will return its corresponding hyperbolic Cosine value.

**Example**

```
self.angle.cosh
```

**2.7.2.1.22.10 Exp**

<b>Source</b>	Double
<b>Parameters</b>	Double power

<b>Result</b>	Double
---------------	--------

**Description**

Returns the source raised to the specified power.

**Example**

```
2 . Exp ( 8 )
```

Returns 256.

**2.7.2.1.22.11 Floor**

<b>Source</b>	Double
<b>Parameters</b>	
<b>Result</b>	Double

**Description**

Rounds the source down to the closest whole number.

**Notes**

An overload also exists for Decimal.

**2.7.2.1.22.12 IsInfinity**

<b>Source</b>	Single
<b>Parameters</b>	
<b>Result</b>	Boolean

**Description**

Returns a value indicating whether the specified number evaluates to negative or positive infinity.

**Notes**

An additional overload exists for Double.

**2.7.2.1.22.13 IsNaN**

<b>Source</b>	Single
<b>Parameters</b>	
<b>Result</b>	Boolean

**Description**

Returns a value indicating whether the specified number evaluates to not a number (NaN).

**Notes**

An additional overload exists for Double.

**2.7.2.1.22.14 IsNegativeInfinity**

<b>Source</b>	Single
<b>Parameters</b>	
<b>Result</b>	Boolean

**Description**

Returns a value indicating whether the specified number evaluates to negative infinity.

**Notes**

An additional overload exists for Double.

**2.7.2.1.22.15 IsPositiveInfinity**

<b>Source</b>	Single
<b>Parameters</b>	
<b>Result</b>	Boolean

**Description**

Returns a value indicating whether the specified number evaluates to positive infinity.

**Notes**

An additional overload exists for Double.

**2.7.2.1.22.16 Log**

<b>Source</b>	Double
<b>Parameters</b>	
<b>Result</b>	Double

**Description**

Returns the natural (base e) logarithm of a specified number.

<b>Source</b>	Double
<b>Parameters</b>	Double newBase
<b>Result</b>	Double

**Description**

Returns the natural (base e) logarithm of a specified number in a specified base.

**2.7.2.1.22.17 Log10**

<b>Source</b>	Double
<b>Parameters</b>	
<b>Result</b>	Double

**Description**

Returns the base 10 logarithm of a specified number.

**2.7.2.1.22.18 Max**

<b>Source</b>	Int32
<b>Parameters</b>	Int32 comparison
<b>Result</b>	Int32

**Description**

Compares the source with the parameter and returns the greater of the two values.

**Example**

```
1 . max ( 2 )
```

**Notes**

Additional overloads exist for SByte, Byte, Int16, UInt16, UInt32, Int64, UInt64, Single, Double, and Decimal.

**2.7.2.1.22.19 Min**

<b>Source</b>	Int32
<b>Parameters</b>	Int32 comparison
<b>Result</b>	Int32

**Description**

Compares the source with the parameter and returns the lesser of the two values.

**Example**

```
2.min(1)
```

**Notes**

Additional overloads exist for SByte, Byte, Int16, UInt16, UInt32, Int64, UInt64, Single, Double, and Decimal.

**2.7.2.1.22.20 Negate**

<b>Source</b>	Decimal
<b>Parameters</b>	
<b>Result</b>	Decimal

**Description**

Returns the negated value of the source. A positive value will be result in a negative, and a negative in a positive.

**2.7.2.1.22.21 Pow**

<b>Source</b>	Double
<b>Parameters</b>	Double exponent
<b>Result</b>	Decimal

**Description**

Returns the source raised to the power of the specified exponent.

**2.7.2.1.22.22 Remainder**

<b>Source</b>	Decimal
<b>Parameters</b>	Decimal divisor
<b>Result</b>	Decimal

**Description**

Divides the source by the specified divisor and returns the remainder.



**2.7.2.1.22.23 Round**

<b>Source</b>	Decimal
<b>Parameters</b>	
<b>Result</b>	Decimal

**Description**

Returns the result of rounding the source to the nearest whole number.

<b>Source</b>	Decimal
<b>Parameters</b>	Int32 fractionalDigits
<b>Result</b>	Decimal

**Description**

Returns the result of rounding the source to the specified number of fractional digits.

**Notes**

An overload also exists for Double.

**2.7.2.1.22.24 Sin**

<b>Source</b>	Double
<b>Parameters</b>	
<b>Result</b>	Double

**Description**

Using the source as an angle this operation will return its corresponding Sine value.

**Example**

```
self.angle.sin
```

**2.7.2.1.22.25 Sinh**

<b>Source</b>	Double
<b>Parameters</b>	
<b>Result</b>	Double

**Description**

Using the source as an angle this operation will return its corresponding hyperbolic Sine value.

**Example**

```
self.angle.sinh
```

**2.7.2.1.22.26 Sqrt**

<b>Source</b>	Double
<b>Parameters</b>	
<b>Result</b>	Double

**Description**

Returns the square root of the source.

**2.7.2.1.22.27 Tan**

<b>Source</b>	Double
<b>Parameters</b>	
<b>Result</b>	Double

**Description**

Using the source as an angle this operation will return its corresponding Tangent value.

**Example**

```
self.angle.tan
```

**2.7.2.1.22.28 Tanh**

<b>Source</b>	Double
<b>Parameters</b>	
<b>Result</b>	Double

**Description**

Using the source as an angle this operation will return its corresponding hyperbolic Tangent value.

**Example**

```
self.angle.tanh
```

### 2.7.2.1.22.29 Truncate

<b>Source</b>	Decimal
<b>Parameters</b>	
<b>Result</b>	Decimal

#### Description

Returns the whole number part of the source, any fractional part is discarded.

#### Notes

An overload exists for Double.

### 2.7.2.1.23 MaxLength

<b>Source</b>	<String attribute>
<b>Parameters</b>	
<b>Result</b>	Int32

#### Description

Returns the maximum value length a string property may hold, as defined in the model.

#### Example

```
Person instance = new Person(EcoSpace);
int maxLength = ecoSpace.Ocl.Evaluate(instance.AsIObject(),
"self.firstName.maxLength").GetValue<int>();
```

#### Notes

This OCL operation can be useful for ensuring maximum lengths are not exceeded in your UI. Simply add an additional column to your ECO handle for each string attribute on your class with the expression "self.<member name>.maxLength" - it is then possible to databind the MaxLength of a TextBox for example to this value rather than hard-coding it and having to manually update if the model changes.

### 2.7.2.1.24 ModifiedSinceTimeStamp

<b>Source</b>	<Instance>
<b>Parameters</b>	Int32 timeStamp
<b>Result</b>	Boolean

**Description**

Returns True if the instance has been modified on or after the specified time stamp, otherwise False. This OCL operation supports the object versioning mechanism.

**2.7.2.1.25 NewGuid**

<b>Source</b>	<Static operation> Guid
<b>Parameters</b>	
<b>Result</b>	Guid

**Description**

Returns a new Guid.

**Example**

```
'here is a new guid'+Guid.NewGuid.asString
```

**2.7.2.1.26 ObjectFromExternalId**

<b>Source</b>	<Type>
<b>Parameters</b>	string externalId
<b>Result</b>	<Instance>

**Description**

Returns the object with the given persistent ID. The ID consists of the class type and the primary key of the object, and may be obtained using the ExternalId (see page 84) operation. This operation is useful in ASP.NET applications where the ID of an object is passed as a query parameter in a URL.

**Example**

```
Person.objectFromExternalId(var_ExternalId)
```

This expression is very useful to use in an EcoDataSource on an ASP.Net page where the variable var\_ExternalId can be bound to a parameter in the URL.

**2.7.2.1.27 ObjectTimeStamp**

<b>Source</b>	<Instance>
<b>Parameters</b>	
<b>Result</b>	Int32

**Description**

Returns the version time stamp of the source. This value will typically be Int32.MaxValue, except when you use a historical object version as the source.

### 2.7.2.1.28 OclAsType

<b>Source</b>	<Instance>
<b>Parameters</b>	
<b>Result</b>	<Instance>

#### Description

Casts the source to the specified sub-class.

#### Example

```
BaseClass.allInstances->first.oclAsType(SubClass).attributeOnSubClass
```

<b>Source</b>	Collection(<Instance>)
<b>Parameters</b>	
<b>Result</b>	Collection(<Instance>)

#### Description

Casts the source collection to the specified sub-class.

#### Example

```
BaseClass.allInstances.oclAsType(SubClass).attributeOnSubClass
```

#### Notes

An InvalidCastException will be thrown if the source cannot be cast to the specified type.

### 2.7.2.1.29 OclIsKindOf

<b>Source</b>	<Instance>
<b>Parameters</b>	<Type>
<b>Result</b>	Boolean

#### Description

Returns True if the source may be type cast to the type specified as a parameter.

**Example**

```
//True. SubClass may be cast to a BaseClass.
bool subClassIsKindOfBaseClass = EcoSpace.Ocl.Evaluate(
    subClass.AsIObject(),
    "self.oclIsKindOf(BaseClass)").GetValue<bool>();

//False. BaseClass may not be cast to a SubClass
bool baseClassIsKindOfSubClass = EcoSpace.Ocl.Evaluate(
    baseClass.AsIObject(),
    "self.oclIsKindOf(SubClass)").GetValue<bool>();
```

**2.7.2.1.30 OclIsTypeOf**

<b>Source</b>	<Instance>
<b>Parameters</b>	<Type>
<b>Result</b>	Boolean

**Description**

Returns True if the source is an instance of the type specified by the parameter.

**Example**

```
//True. subClass is an instance of the SubClass type.
bool subClassIsTypeOfSubClass = EcoSpace.Ocl.Evaluate(
    subClass.AsIObject(),
    "self.oclIsTypeOf(SubClass)").GetValue<bool>();

//False. subClass is not an instance of the BaseClass type, despite being descended
from BaseClass.
bool subClassIsTypeOfBaseClass = EcoSpace.Ocl.Evaluate(
    subClass.AsIObject(),
    "self.oclIsTypeOf(BaseClass)").GetValue<bool>();
```

**2.7.2.1.31 Parse**

<b>Source</b>	<Static operation> <Type>
<b>Parameters</b>	String value
<b>Result</b>	<Instance of Type>

**Description**

The Parse operation executes the static Parse(String) method on the specified type. This operation is available on all standard .NET types that implement the method.

**Example**

```
DateTime.Parse('2001-01-31')
DateTime.Parse('2001-01-31T12:30:59')
Byte.Parse('255')
```

### 2.7.2.1.32 SafeCast

<b>Source</b>	<Instance>
<b>Parameters</b>	<Type>
<b>Result</b>	<Instance>

#### Description

Casts the source to the specified type. If the source is not compatible with the type the operation will silently fail and return null/nil.

#### Example

```
var baseClass = new BaseClass(EcoSpace);
var subClass =
  EcoSpace.Ocl.Evaluate(
    baseClass.AsIObject(),
    "self.safeCast(SubClass)".GetValue<SubClass>());
```

Results in subClass being null.

<b>Source</b>	Collection(<Instance>)
<b>Parameters</b>	<Type>
<b>Result</b>	Collection(<Instance>)

#### Description

Casts each instance in the source collection to the specified type. If the source is not compatible with the type the operation will silently fail and add null/nil for the individual instance before continuing with the next instance in the source.

#### Example

```
var baseClass = new BaseClass(EcoSpace);
var subClass = new SubClass(EcoSpace);
var subClassListCount =
  EcoSpace.Ocl.Evaluate(
    baseClass.AsIObject(),
    "BaseClass.allInstances.safeCast(SubClass)".
    .GetAsCollection().Count);
```

Results in subClassListCount holding the value 2, the resulting collection holds a valid object and a nil reference.

### 2.7.2.1.33 State machine operations

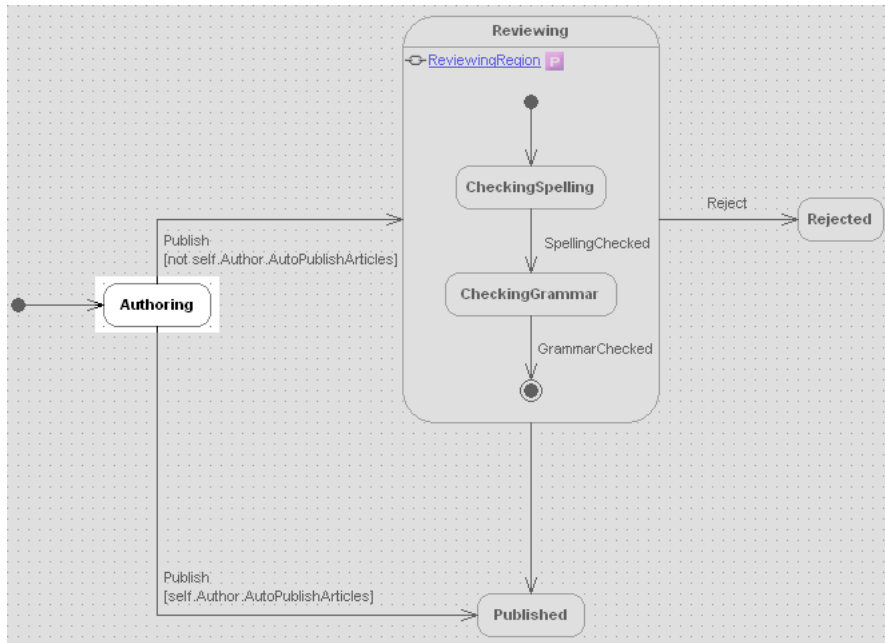
#### 2.7.2.1.33.1 OclGetStates

<b>Source</b>	<Instance>
<b>Parameters</b>	
<b>Result</b>	Collection(String)

**Description**

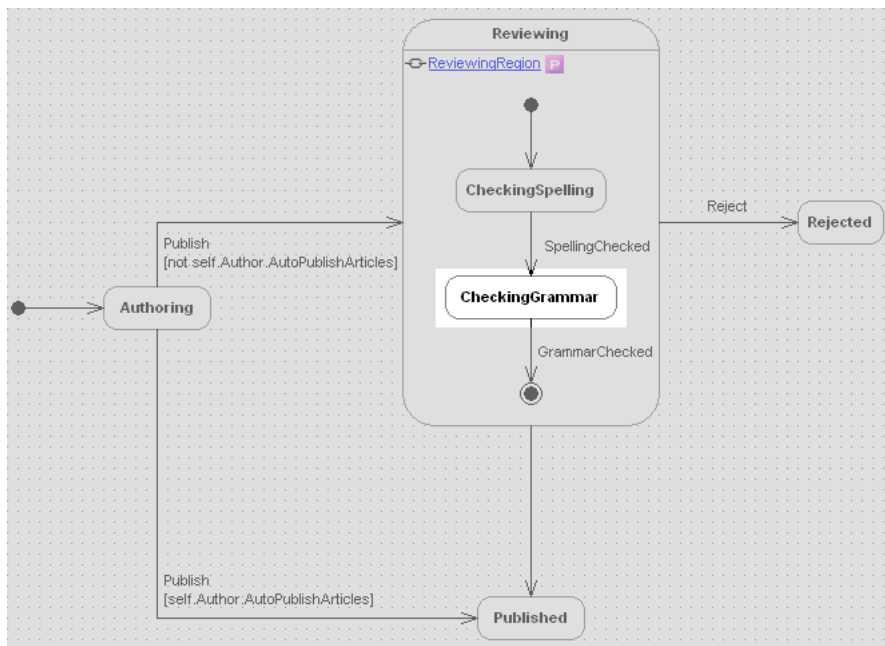
Given a source instance this operation will return a string for each state the object is in.

**Example**



**Result**

- Authoring



**Result**

- Reviewing.CheckingGrammar



If the state machine contains parallel states, the result from this operation can contain more than one state.

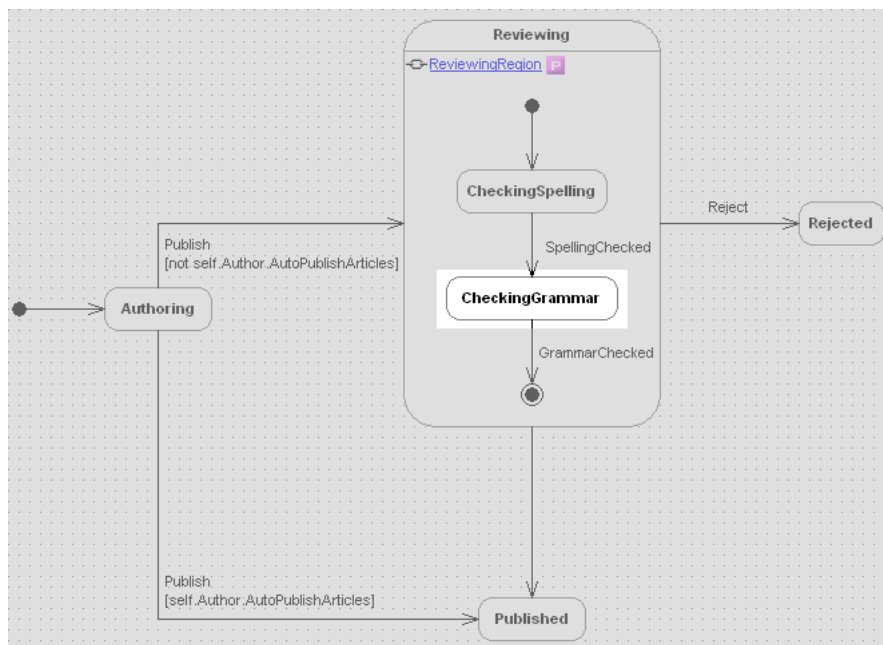
### 2.7.2.1.33.2 OclGetTriggers

<b>Source</b>	<Instance>
<b>Parameters</b>	
<b>Result</b>	Collection(String)

#### Description

Given a source instance this operation will return a string for each available trigger depending on its current state. Guard expressions are not evaluated in order to include/exclude triggers from the result.

#### Example



#### Result

- GrammarChecked
- Reject

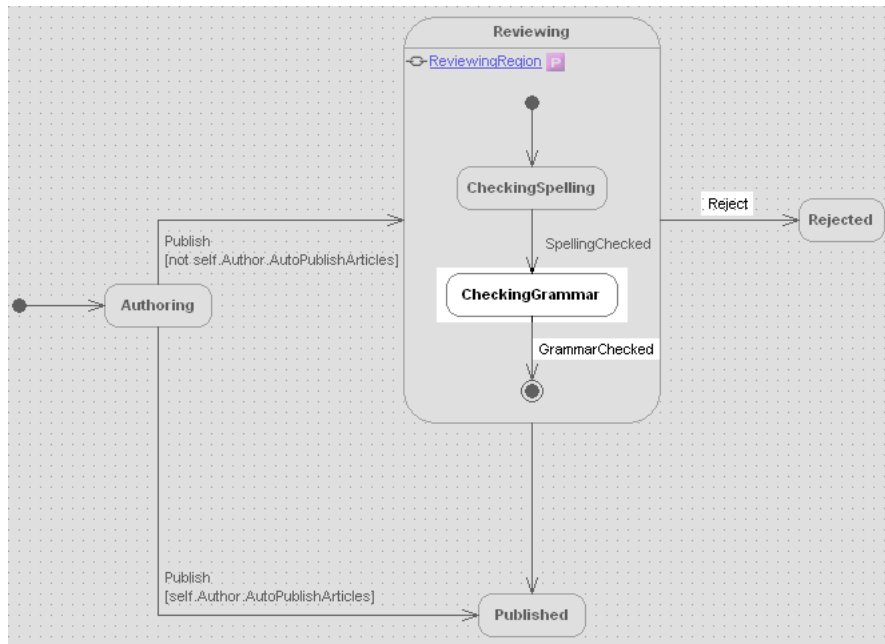
### 2.7.2.1.33.3 OclIsInState

<b>Source</b>	<Instance>
<b>Parameters</b>	EnumLiteral StateName
<b>Result</b>	Boolean

#### Description

Given a source instance and the name of a state this operation will return True or False depending on whether or not the instance's state machine is in that state.

**Example**



```

//True
bool isReviewing =
    EcoSpace.Ocl.Evaluate(
        article.AsIObject(),
        "self.oclIsInState(#Reviewing)").GetValue<bool>();

//True
bool isCheckingGrammar =
    EcoSpace.Ocl.Evaluate(
        article.AsIObject(),
        "self.oclIsInState(#CheckingGrammar)").GetValue<bool>();

//False
bool isPublished =
    EcoSpace.Ocl.Evaluate(
        article.AsIObject(),
        "self.oclIsInState(#Published)").GetValue<bool>();
    
```

**2.7.2.1.34 String operations**

**2.7.2.1.34.1 Chars**

<b>Source</b>	String
<b>Parameters</b>	Int32 index
<b>Result</b>	Char

**Description**

Returns the character at the specified index of the source. The index is zero based.

**Example**

```
'Hello'.chars(0)
```

**2.7.2.1.34.2 ClrSubstring**

<b>Source</b>	String
<b>Parameters</b>	Int32 firstIndex
<b>Result</b>	String

**Description**

Uses the CLR String.Substring routine to return a substring of the source.

**Example**

```
'Hello'.substring(2)
```

Returns "llo".

<b>Source</b>	String
<b>Parameters</b>	Int32 firstIndex Int32 length
<b>Result</b>	String

**Description**

Uses the CLR String.Substring routine to return a substring of the source.

**Example**

```
'Hello'.substring(2, 2)
```

Returns "ll".

**2.7.2.1.34.3 Concat**

<b>Source</b>	String
<b>Parameters</b>	String value
<b>Result</b>	String

**Description**

Returns the result of appending the parameter to the source.

**Example**

```
'Hello'.concat(' World')
```

**Notes**

Additional overloads exist to enable you to specify between one and three values to append to the end of the source.

**2.7.2.1.34.4 Contains**

<b>Source</b>	String
<b>Parameters</b>	String value
<b>Result</b>	Boolean

**Description**

Returns True if the text specified exists within the source.

**Example**

```
'Hello'.contains('llo')
```

**2.7.2.1.34.5 EndsWith**

<b>Source</b>	String
<b>Parameters</b>	String value
<b>Result</b>	Boolean

**Description**

Returns True if the source ends with the string specified.

**Example**

```
'Hello'.endsWith('lo')
```

Returns True.

### 2.7.2.1.34.6 Format

<b>Source</b>	String
<b>Parameters</b>	<ul style="list-style-type: none"> <li>• String source</li> <li>• Object value</li> </ul>
<b>Result</b>	String

#### Description

Uses the .NET String.Format method on the source to produce a new string. The "value" parameter may be repeated multiple times.

#### Example

```
String.format('The time now is {0:g}', DateTime.Now)
```

### 2.7.2.1.34.7 GetNumericValue

<b>Source</b>	Char
<b>Parameters</b>	
<b>Result</b>	Double

#### Description

Converts the specified numeric Unicode character to a double-precision floating point number.

### 2.7.2.1.34.8 IndexOf

<b>Source</b>	String
<b>Parameters</b>	Char character
<b>Result</b>	Int32

#### Description

Uses the CLR String.Substring routine. Returns the index of the first occurrence of the specified Unicode character in the source string.

#### Example

```
'Hello'.indexOf('e')
```

<b>Source</b>	String
<b>Parameters</b>	String subString

<b>Result</b>	Int32
---------------	-------

**Description**

Uses the CLR String.Substring routine. Returns the index of the first occurrence of the specified sub string in the source string.

**Example**

'Hello'.indexOf('llo')

<b>Source</b>	String
<b>Parameters</b>	<ul style="list-style-type: none"> <li>• Char character</li> <li>• Int32 startPosition</li> </ul>
<b>Result</b>	Int32

**Description**

Uses the CLR String.Substring routine. Returns the index of the first occurrence of the specified Unicode character in the source string. The search starts at the specified zero-based character position.

**Example**

'Hello hello'.indexOf('l', 4)

<b>Source</b>	String
<b>Parameters</b>	<ul style="list-style-type: none"> <li>• String subString</li> <li>• Int32 startPosition</li> </ul>
<b>Result</b>	Int32

**Description**

Uses the CLR String.Substring routine. Returns the index of the first occurrence of the specified sub string in the source string. The search starts at the specified zero-based character position.

**Example**

'Hello hello'.indexOf('ll', 4)

<b>Source</b>	String
<b>Parameters</b>	<ul style="list-style-type: none"> <li>• Char character</li> <li>• Int32 startPosition</li> <li>• Int32 subStringLength</li> </ul>
<b>Result</b>	Int32

**Description**

Uses the CLR String.Substring routine. Returns the index of the first occurrence of the specified Unicode character in the source string. The search starts at the specified zero-based character position and only inspects the specified number of characters.

**Example**

```
'Hello hello'.indexOf('l', 5, 3)
```

<b>Source</b>	String
<b>Parameters</b>	<ul style="list-style-type: none"> <li>• String subString</li> <li>• Int32 startPosition</li> <li>• Int32 subStringLength</li> </ul>
<b>Result</b>	Int32

**Description**

Uses the CLR String.Substring routine. Returns the index of the first occurrence of the specified sub string in the source string. The search starts at the specified zero-based character position and only inspects the specified number of characters.

**Example**

```
'Hello hello'.indexOf('ll', 5, 5)
```

**2.7.2.1.34.9 Insert**

<b>Source</b>	String
<b>Parameters</b>	<ul style="list-style-type: none"> <li>• Int32 startPosition</li> <li>• String stringToInsert</li> </ul>
<b>Result</b>	String

**Description**

Returns a new string by inserting the parameter string into the source at the specified zero-based index.

**Example**

```
'Helo'.insert(2, 'l')
```

### 2.7.2.1.34.10 IsControl

<b>Source</b>	Char
<b>Parameters</b>	
<b>Result</b>	Boolean

#### Description

Indicates whether a specified Unicode character is categorized as a control character.

#### Example

```
self.firstName.chars(0).isControl
```

<b>Source</b>	<Static operation> Char
<b>Parameters</b>	<ul style="list-style-type: none"> <li>• String value</li> <li>• Int32 index</li> </ul>
<b>Result</b>	Boolean

#### Description

Indicates whether the character at the specified position in a specified string is categorized as a control character.

#### Example

```
Char.isControl('Some string with a control character', 2)
```

### 2.7.2.1.34.11 IsDigit

<b>Source</b>	Char
<b>Parameters</b>	
<b>Result</b>	Boolean

#### Description

Indicates whether a Unicode character is categorized as a decimal digit.

#### Example

```
self.dateOfBirth.asString.chars(2).isDigit
```



<b>Source</b>	<Static operation> Char
<b>Parameters</b>	<ul style="list-style-type: none"> <li>• String value</li> <li>• Int32 index</li> </ul>
<b>Result</b>	Boolean

**Description**

Indicates whether the character at the specified position in a specified string is categorized as a decimal digit.

**Example**

```
Char.IsDigit('ABC9', 3)
```

**2.7.2.1.34.12 IsHighSurrogate**

<b>Source</b>	Char
<b>Parameters</b>	
<b>Result</b>	Boolean

**Description**

Indicates whether the specified Char object is a high surrogate.

**Example**

```
self.firstName.chars(0).isHighSurrogate
```

<b>Source</b>	<Static operation> Char
<b>Parameters</b>	<ul style="list-style-type: none"> <li>• String value</li> <li>• Int32 index</li> </ul>
<b>Result</b>	Boolean

**Description**

Indicates whether the Char at the specified position in a string is a high surrogate.

**Example**

```
Char.IsHighSurrogate('Hello', 0)
```

**2.7.2.1.34.13 IsLetter**

<b>Source</b>	Char
<b>Parameters</b>	
<b>Result</b>	Boolean

**Description**

Indicates whether a Unicode character is categorized as an alphabetic letter.

**Example**

```
self.firstName.chars(0).isLowSurrogate
```

<b>Source</b>	<Static operation> Char
<b>Parameters</b>	<ul style="list-style-type: none"> <li>• String value</li> <li>• Int32 index</li> </ul>
<b>Result</b>	Boolean

**Description**

Indicates whether the character at the specified position in a specified string is categorized as an alphabetic character.

**Example**

```
Char.isLetter('0123A', 4)
```

**2.7.2.1.34.14 IsLetterOrDigit**

<b>Source</b>	Char
<b>Parameters</b>	
<b>Result</b>	Boolean

**Description**

Indicates whether a Unicode character is categorized as an alphabetic letter or a decimal digit.

**Example**

```
self.firstName.chars(0).isLetterOrDigit
```

<b>Source</b>	<Static operation> Char
<b>Parameters</b>	<ul style="list-style-type: none"> <li>• String value</li> <li>• Int32 index</li> </ul>
<b>Result</b>	Boolean

**Description**

Indicates whether the character at the specified position in a specified string is categorized as an alphabetic character or a decimal digit.

**Example**

```
Char.isLetterOrDigit('-AB12-', 0)
```

**2.7.2.1.34.15 IsLower**

<b>Source</b>	Char
<b>Parameters</b>	
<b>Result</b>	Boolean

**Description**

Indicates whether the specified Unicode character is categorized as a lowercase letter.

**Example**

```
self.firstName.chars(0).isLower
```

<b>Source</b>	<Static operation> Char
<b>Parameters</b>	<ul style="list-style-type: none"> <li>• String value</li> <li>• Int32 index</li> </ul>
<b>Result</b>	Boolean

**Description**

Indicates whether the character at the specified position in a specified string is categorized as a lowercase letter.

**Example**

```
Char.isLower('Lower', 1)
```

**2.7.2.1.34.16 IsLowSurrogate**

<b>Source</b>	Char
<b>Parameters</b>	
<b>Result</b>	Boolean

**Description**

Indicates whether the specified Char is a low surrogate.

**Example**

```
self.firstName.chars(0).isLowSurrogate
```

<b>Source</b>	<Static operation> Char
<b>Parameters</b>	<ul style="list-style-type: none"> <li>• String value</li> <li>• Int32 index</li> </ul>
<b>Result</b>	Boolean

**Description**

Indicates whether the Char at the specified position in a string is a low surrogate.

**Example**

```
Char.isLowSurrogate('Hello', 0)
```

**2.7.2.1.34.17 IsNormalized**

<b>Source</b>	String
<b>Parameters</b>	
<b>Result</b>	Boolean

**Description**

Indicates whether this string is in a particular Unicode normalization form.

**2.7.2.1.34.18 IsNullOrEmpty**

<b>Source</b>	String
<b>Parameters</b>	
<b>Result</b>	Boolean

**Description**

Indicates whether the specified String object is a null reference or an empty string.

**2.7.2.1.34.19 IsNumber**

<b>Source</b>	Char
<b>Parameters</b>	
<b>Result</b>	Boolean

**Description**

Indicates whether a Unicode character is categorized as a number.

**Example**

```
self.firstName.chars(0).isNumber
```

<b>Source</b>	<Static operation> Char
<b>Parameters</b>	<ul style="list-style-type: none"> <li>• String value</li> <li>• Int32 index</li> </ul>
<b>Result</b>	Boolean

**Description**

Indicates whether the character at the specified position in a specified string is categorized as a number.

**Example**

```
Char.isNumber('123', 0)
```

**2.7.2.1.34.20 IsPunctuation**

<b>Source</b>	Char
<b>Parameters</b>	
<b>Result</b>	Boolean

**Description**

Indicates whether a Unicode character is categorized as a punctuation mark.

**Example**

```
self.firstName.chars(0).isPunctuation
```

<b>Source</b>	<Static operation> Char
<b>Parameters</b>	<ul style="list-style-type: none"> <li>• String value</li> <li>• Int32 index</li> </ul>
<b>Result</b>	Boolean

**Description**

Indicates whether the character at the specified position in a specified string is categorized as a punctuation mark.

**Example**

```
Char.isPunctuation('Hello world!', 11)
```

**2.7.2.1.34.21 IsSeparator**

<b>Source</b>	Char
<b>Parameters</b>	
<b>Result</b>	Boolean

**Description**

Indicates whether a Unicode character is categorized as a separator character.

**Example**

```
self.firstName.chars(0).isSeparator
```

<b>Source</b>	<Static operation> Char
<b>Parameters</b>	<ul style="list-style-type: none"> <li>• String value</li> <li>• Int32 index</li> </ul>
<b>Result</b>	Boolean

**Description**

Indicates whether the character at the specified position in a specified string is categorized as a separator character.

**Example**

```
Char.isSeparator('\t', 0)
```

**2.7.2.1.34.22 IsSurrogate**

<b>Source</b>	Char
<b>Parameters</b>	
<b>Result</b>	Boolean

**Description**

Indicates whether the specified Char is a low surrogate.

**Example**

```
self.firstName.chars(0).isSurrogate
```

<b>Source</b>	<Static operation> Char
<b>Parameters</b>	<ul style="list-style-type: none"> <li>• String value</li> <li>• Int32 index</li> </ul>
<b>Result</b>	Boolean

**Description**

Indicates whether the Char at the specified position in a string is a low surrogate.

**Example**

```
Char.isSurrogate('Hello', 0)
```

**2.7.2.1.34.23 IsSurrogatePair**

<b>Source</b>	Char
<b>Parameters</b>	
<b>Result</b>	Boolean

**Description**

Indicates whether the two specified Chars form a surrogate pair. .

**Example**

```
self.firstName.chars(0).isSurrogatePair(self.firstName.chars(1))
```

<b>Source</b>	<Static operation> Char
<b>Parameters</b>	<ul style="list-style-type: none"> <li>• String value</li> <li>• Int32 index</li> </ul>
<b>Result</b>	Boolean

**Description**

Indicates whether two adjacent Chars at the specified position form a surrogate pair.

**Example**

```
Char.isSurrogatePair(self.firstName, 0)
```

### 2.7.2.1.34.24 IsSymbol

<b>Source</b>	Char
<b>Parameters</b>	
<b>Result</b>	Boolean

#### Description

Indicates whether a Unicode character is categorized as a symbol.

#### Example

```
self.firstName.chars(0).isSymbol
```

<b>Source</b>	<Static operation> Char
<b>Parameters</b>	<ul style="list-style-type: none"> <li>• String value</li> <li>• Int32 index</li> </ul>
<b>Result</b>	Boolean

#### Description

Indicates whether the character at the specified position in a specified string is categorized as a symbol.

#### Example

```
Char.isSymbol('1 + 1 = 2', 2)
```

### 2.7.2.1.34.25 IsUpper

<b>Source</b>	Char
<b>Parameters</b>	
<b>Result</b>	Boolean

#### Description

Indicates whether the specified Unicode character is categorized as an uppercase letter.

#### Example

```
self.firstName.chars(0).isUpper
```



<b>Source</b>	<Static operation> Char
<b>Parameters</b>	<ul style="list-style-type: none"> <li>• String value</li> <li>• Int32 index</li> </ul>
<b>Result</b>	Boolean

**Description**

Indicates whether the character at the specified position in a specified string is categorized as a uppercase letter.

**Example**

```
Char.ToUpper('Upper', 0)
```

**2.7.2.1.34.26 IsWhiteSpace**

<b>Source</b>	Char
<b>Parameters</b>	
<b>Result</b>	Boolean

**Description**

Indicates whether the specified Unicode character is categorized as a white space letter.

**Example**

```
self.firstName.chars(0).isWhiteSpace
```

<b>Source</b>	<Static operation> Char
<b>Parameters</b>	<ul style="list-style-type: none"> <li>• String value</li> <li>• Int32 index</li> </ul>
<b>Result</b>	Boolean

**Description**

Indicates whether the character at the specified position in a specified string is categorized as a white space letter.

**Example**

```
Char.IsWhiteSpace('Hello world', 5)
```

### 2.7.2.1.34.27 LastIndexOf

<b>Source</b>	String
<b>Parameters</b>	String value
<b>Result</b>	Boolean

#### Description

Reports the index position of the last occurrence of a specified string within the source string.

#### Example

```
self.firstName.lastIndexOf('.')
```

<b>Source</b>	String
<b>Parameters</b>	<ul style="list-style-type: none"> <li>• String value</li> <li>• Int32 position</li> </ul>
<b>Result</b>	Boolean

#### Description

Reports the index position of the last occurrence of a specified string within the source string. This overload searches a sub-string of the source, the substring will be the first character (position 0) up to and including the position specified.

#### Example

```
self.firstName.lastIndexOf('.', 5)
```

<b>Source</b>	String
<b>Parameters</b>	<ul style="list-style-type: none"> <li>• String value</li> <li>• Int32 startIndex</li> <li>• Int32 length</li> </ul>
<b>Result</b>	Boolean

#### Description

Reports the index position of the last occurrence of a specified string within the source string. This overload searches a sub-string of the source, the substring will be the character specified by the startIndex with the specified length.

**Example**

```
self.firstName.lastIndexOf('.', 3, 10)
```

**2.7.2.1.34.28 Normalize**

<b>Source</b>	String
<b>Parameters</b>	
<b>Result</b>	String

**Description**

Returns a new string whose binary representation is in a particular Unicode normalization form.

**2.7.2.1.34.29 Pad**

<b>Source</b>	String
<b>Parameters</b>	<ul style="list-style-type: none"> <li>• Int32 newLength</li> <li>• String paddingString</li> </ul>
<b>Result</b>	String

**Description**

Returns a new string based on the source with a guaranteed minimum length. If the source is shorter than the specified new length it will be left-padded with the specified text, if it is already equal to or longer than the specified new length the result will be the same as the source.

**Example**

```
//returns 21212Hello
'Hello'.pad(10, '12')
```

**2.7.2.1.34.30 PadLeft**

<b>Source</b>	String
<b>Parameters</b>	Int32 newLength
<b>Result</b>	String

**Description**

Returns a new string based on the source with a guaranteed minimum length. If the source is shorter than the specified new length the left of the string be padded with spaces, if it is already equal to or longer than the specified new length the result will be the same as the source.

**Example**

```
'Hello'.padLeft(6)
```

Returns " Hello"

```
'Hello'.padLeft(3)
```

Returns "Hello"

<b>Source</b>	String
<b>Parameters</b>	Int32 newLength Char paddingCharacter
<b>Result</b>	String

**Description**

Returns a new string based on the source with a guaranteed minimum length. If the source is shorter than the specified new length the left of the string be padded with the specified character, if it is already equal to or longer than the specified new length the result will be the same as the source.

**Example**

```
'Hello'.padLeft(10, 'a'.chars(0))
```

Returns "aaaaaHello"

**2.7.2.1.34.31 PadRight**

<b>Source</b>	String
<b>Parameters</b>	Int32 newLength
<b>Result</b>	String

**Description**

Returns a new string based on the source with a guaranteed minimum length. If the source is shorter than the specified new length the right of the string be padded with spaces, if it is already equal to or longer than the specified new length the result will be the same as the source.

**Example**

```
'Hello'.padRight(6)
```

Returns "Hello "

```
'Hello'.padRight(3)
```

Returns "Hello"

<b>Source</b>	String
<b>Parameters</b>	Int32 newLength Char paddingCharacter
<b>Result</b>	String

**Description**

Returns a new string based on the source with a guaranteed minimum length. If the source is shorter than the specified new length the right of the string be padded with the specified character, if it is already equal to or longer than the specified new length the result will be the same as the source.

**Example**

```
'Hello'.padRight(10, 'a'.chars(0))
```

Returns "Helloaaaaa"

**2.7.2.1.34.32 PostPad**

<b>Source</b>	String
<b>Parameters</b>	Int32 newLength String paddingString
<b>Result</b>	String

**Description**

Returns a new string based on the source with a guaranteed minimum length. If the source is shorter than the specified new length the right of the string be padded with the specified padding string, truncating the result if necessary. If the source is already equal to or longer than the specified new length the result will be the same as the source.

**2.7.2.1.34.33 RegExpMatch**

<b>Source</b>	String
<b>Parameters</b>	String pattern
<b>Result</b>	Boolean

**Description**

Evaluates the specified pattern against the source and returns whether or not the source matches the pattern.

For information of the syntax for regular expression, please refer to the .net framework documentation.

**2.7.2.1.34.34 Remove**

<b>Source</b>	String
<b>Parameters</b>	Int32 firstCharToRemove
<b>Result</b>	String

**Description**

Removes the end of the source string from the firstCharToRemove onwards and returns the result.

**Example**

```
'Hello World!'.Remove(5)
```

Returns "Hello"

<b>Source</b>	String
<b>Parameters</b>	Int32 firstCharToRemove Int32 numberOfCharsToRemove

<b>Result</b>	String
---------------	--------

**Description**

Removes the specified number of characters from the source string starting from firstCharToRemove.

**Example**

```
'Hello World!'.Remove(5, 6)
```

Returns "Hello!"

**2.7.2.1.34.35 Replace**

<b>Source</b>	String
<b>Parameters</b>	String textToFind String textToReplace
<b>Result</b>	String

**Description**

Returns a string based on the source but with all occurrences of textToFind replaced with textToReplace.

**Example**

```
'Hello World!'.Replace('World', 'mother')
```

Returns "Hello mother!"

**2.7.2.1.34.36 StartsWith**

<b>Source</b>	String
<b>Parameters</b>	String value
<b>Result</b>	Boolean

**Description**

Returns True if the text specified starts with the value specified.

**Example**

```
'Hello'.startsWith('He')
```

**2.7.2.1.34.37 StrToDate**

<b>Source</b>	String
<b>Parameters</b>	
<b>Result</b>	DateTime

**Description**

Uses DateTime.Parse to convert the source to a DateTime.

**2.7.2.1.34.38 StrToDateTime**

<b>Source</b>	String
<b>Parameters</b>	
<b>Result</b>	DateTime

**Description**

Uses DateTime.Parse to convert the source to a DateTime.

**2.7.2.1.34.39 StrToInt**

<b>Source</b>	String
<b>Parameters</b>	
<b>Result</b>	Int32

**Description**

Uses Int32.Parse to convert the source to an Int32.

**2.7.2.1.34.40 StrToTime**

<b>Source</b>	String
<b>Parameters</b>	
<b>Result</b>	TimeSpan

**Description**

Uses TimeSpan.Parse to convert the source to a TimeSpan.

**2.7.2.1.34.41 SubString**

<b>Source</b>	String
<b>Parameters</b>	Int32 startIndex Int32 length
<b>Result</b>	String

**Description**

Returns a sub string of the source.

**2.7.2.1.34.42 ToLower (Char)**

<b>Source</b>	Char
<b>Parameters</b>	
<b>Result</b>	Char

**Description**

Converts a char value to lower case.

**Example**

```
self.productCode.chars(1).toLowerCase
```

**2.7.2.1.34.43 ToUpper (Char)**

<b>Source</b>	Char
<b>Parameters</b>	
<b>Result</b>	Char

**Description**

Converts a char value to upper case.

**Example**

```
self.productCode.chars(1).toUpperCase
```

**2.7.2.1.34.44 ToLowerInvariant**

<b>Source</b>	String
<b>Parameters</b>	
<b>Result</b>	String

**Description**

Converts a string to lower case using casing rules of the invariant culture.

**Example**

```
self.productCode.toLowerCaseInvariant
```

**Notes**

An overload exists for Char.

**2.7.2.1.34.45 ToUpperInvariant**

<b>Source</b>	String
<b>Parameters</b>	
<b>Result</b>	String



**Description**

Converts a string to upper case using casing rules of the invariant culture.

**Example**

```
self.productCode.toUpperInvariant
```

**Notes**

An overload exists for Char.

**2.7.2.1.34.46 Trim**

<b>Source</b>	String
<b>Parameters</b>	
<b>Result</b>	String

**Description**

Returns the source with all leading and trailing white space characters removed.

**Example**

```
self.productCode.trim
```

**2.7.2.1.35 SuperTypes**

<b>Source</b>	Type
<b>Parameters</b>	
<b>Result</b>	Collection(String)

**Description**

Returns a list of class names, one for each supertype of the source.

**2.7.2.1.36 TimeSpan operations****2.7.2.1.37 TaggedValue**

<b>Source</b>	<Instance>
<b>Parameters</b>	String taggedValueName
<b>Result</b>	String

**Description**

Finds a tagged value with the specified name defined against the instance's class (or one of its super classes) and returns the value defined in the model.

### 2.7.2.1.38 ToByte

<b>Source</b>	Decimal
<b>Parameters</b>	
<b>Result</b>	Byte

#### Description

If the source is within the range `Byte.MinValue .. Byte.MaxValue` the result will be a byte, otherwise a `System.TargetInvocationException` will be thrown (with an inner exception type of `System.OverflowException`). Any fractional value will be discarded.

### 2.7.2.1.39 ToDouble

<b>Source</b>	Decimal
<b>Parameters</b>	
<b>Result</b>	Double

#### Description

If the source is within the range `Double.MinValue .. Double.MaxValue` the result will be a Double, otherwise a `System.TargetInvocationException` will be thrown (with an inner exception type of `System.OverflowException`). Any fractional value will be discarded.

### 2.7.2.1.40 ToInt16

<b>Source</b>	Decimal
<b>Parameters</b>	
<b>Result</b>	Int16

#### Description

If the source is within the range `Int16.MinValue .. Int16.MaxValue` the result will be an Int16, otherwise a `System.TargetInvocationException` will be thrown (with an inner exception type of `System.OverflowException`). Any fractional value will be discarded.

### 2.7.2.1.41 ToInt32

<b>Source</b>	Decimal
<b>Parameters</b>	
<b>Result</b>	Int32

#### Description

If the source is within the range `Int32.MinValue .. Int32.MaxValue` the result will be an Int32, otherwise a `System.TargetInvocationException` will be thrown (with an inner exception type of `System.OverflowException`). Any fractional value will be discarded.

### 2.7.2.1.42 ToInt64

<b>Source</b>	Decimal
<b>Parameters</b>	
<b>Result</b>	Int64

#### Description

If the source is within the range `Int64.MinValue .. Int64.MaxValue` the result will be an `Int64`, otherwise a `System.TargetInvocationException` will be thrown (with an inner exception type of `System.OverflowException`). Any fractional value will be discarded.

### 2.7.2.1.43 ToSByte

<b>Source</b>	Decimal
<b>Parameters</b>	
<b>Result</b>	SByte

#### Description

If the source is within the range `SByte.MinValue .. SByte.MaxValue` the result will be a signed byte, otherwise a `System.TargetInvocationException` will be thrown (with an inner exception type of `System.OverflowException`). Any fractional value will be discarded.

### 2.7.2.1.44 ToSingle

<b>Source</b>	Decimal
<b>Parameters</b>	
<b>Result</b>	Single

#### Description

If the source is within the range `Single.MinValue .. Single.MaxValue` the result will be a `Single`, otherwise a `System.TargetInvocationException` will be thrown (with an inner exception type of `System.OverflowException`). Any fractional value will be discarded.

### 2.7.2.1.45 ToUInt16

<b>Source</b>	Decimal
<b>Parameters</b>	
<b>Result</b>	UInt16

#### Description

If the source is within the range `UInt16.MinValue .. UInt16.MaxValue` the result will be an `UInt16`, otherwise a `System.TargetInvocationException` will be thrown (with an inner exception type of `System.OverflowException`). Any fractional value will be discarded.

### 2.7.2.1.46 ToUInt32

Source	Decimal
Parameters	
Result	UInt32

#### Description

If the source is within the range UInt32.MinValue .. UInt32.MaxValue the result will be an UInt32, otherwise a System.TargetInvocationException will be thrown (with an inner exception type of System.OverflowException). Any fractional value will be discarded.

### 2.7.2.1.47 ToUInt64

Source	Decimal
Parameters	
Result	UInt64

#### Description

If the source is within the range UInt64.MinValue .. UInt64.MaxValue the result will be an UInt64, otherwise a System.TargetInvocationException will be thrown (with an inner exception type of System.OverflowException). Any fractional value will be discarded.

### 2.7.2.1.48 TypeName

Source	<Type>
Parameters	
Result	String

#### Description

Returns the class name of the source type.

### 2.7.2.1.49 Xor

Source	Boolean
Parameters	Boolean comparison
Result	Boolean

#### Description

Returns the result of performing a logical exclusive OR on the source and parameter

#### Example

```
false xor false = false
```

```

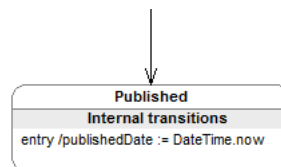
false xor true = true
true xor false = true
true xor true = false

```

## 2.7.3 IActionLanguageService

The Action Language Service is an OCL based expression executor. As well as the OclPs and Ocl commands, which have no side effects, this service is additionally capable of executing statements which alter object state. The Action Language Service is particularly useful when designing state machines. For example when an Article enters a Published state it makes sense to set the article's PublishedDate to the current date and time.

This would be achieved by setting an Entry Action on the Published state with a simple action language expression



Assignments in the action language service are made using the token :=

### 2.7.3.1 Operations support by IActionLanguageService

The IActionLanguageService supports all of the IOclPsService operations (see page 28), all of the IOclServiceOperations (see page 48) operations, and the following additional operations.

#### 2.7.3.1.1 Clear

<b>Source</b>	Collection(<Any>)
<b>Parameters</b>	
<b>Result</b>	

##### Description

Removes all elements from the source collection.

##### Notes

The source must be a member of a modeled class.

```

//Valid
self.orderLines->clear

//Invalid
self.orderLines->select(value < 100)->clear

```

### 2.7.3.1.2 Add

<b>Source</b>	Collection(<Any>)
<b>Parameters</b>	<Any> instanceToAdd
<b>Result</b>	Collection(<Any>)

#### Description

Adds the parameter to the source.

#### Notes

The source must be a member of a modeled class.

```
//Valid
self.orderLines->add(OrderLine.Create)

//Invalid
self.orderLines->select(value < 100)->add(OrderLine.Create)
```

### 2.7.3.1.3 Create

<b>Source</b>	<Type>
<b>Parameters</b>	
<b>Result</b>	<Instance>

#### Description

Creates an instance of the specified modeled class.

#### Example

```
Person.Create
```

### 2.7.3.1.4 Delete

<b>Source</b>	<Type>
<b>Parameters</b>	
<b>Result</b>	Boolean

#### Description

Deletes the specified object instance.

### 2.7.3.1.5 Remove

<b>Source</b>	Collection(<Any>)
<b>Parameters</b>	<Any> instanceToRemove
<b>Result</b>	Collection(<Any>)

#### Description

Removes the specified parameter from the source collection.

### Notes

The source must be a member of a modeled class.

```
//Valid
self.orderLines->remove(self.orderLines->last)

//Invalid
self.orderLines->select(value < 100)->remove(self.orderLines->last)
```

### 2.7.3.1.6 RemoveAt

<b>Source</b>	Collection(<Any>)
<b>Parameters</b>	<Any> indexToRemove
<b>Result</b>	Collection(<Any>)

### Description

Removes the item at the specified index from the source collection.

### Notes

The source must be a member of a modeled class.

```
//Valid
self.orderLines->removeAt(1)

//Invalid
self.orderLines->select(value < 100)->removeAt(1)
```

## 2.7.4 ITypeService

The type service allows you to perform various functions on the OCL evaluator, such as determining the result type of an expression or registering new OCL operations. The members of the ITypeService are implemented directly on each of the three OCL services. So instead of obtaining an ITypeService from the EcoSpace you would use the equivalent method on the OCL service.

### Determining if the result of an expression is read-only

When evaluating an OCL expression the result of that evaluation is either mutable or read-only. A result is mutable if the expression results in a domain member (a member of a class) and that member is not read-only.

```
//Mutable, this is a domain member (modeled)
self.firstName

//Read-only, because this is an expression derived from a domain member
self.firstName.toUpper

//Read-only, because the value is assigned by the DB
//self.uniqueID
```

To determine whether or not an expression is readonly use the ExpressionIsReadOnly method on the appropriate OCL service.

```
IClassifier classifier = EcoSpace.TypeSystem.GetClassByType(typeof(Person));
bool readonly = EcoSpace.Ocl.ExpressionIsReadOnly("person", classifier,
false).GetValue<bool>();
```

### Determining model information from an expression

If the expression results in a domain element (Person.FirstName for example) it is possible to get information about the member as follows

```
IClassifier classifier = EcoSpace.TypeSystem.GetClassByType(typeof(Person));
IStructuralFeature modelInformation = EcoSpace.Ocl.ExpressionModelInfo("self.firstName",
classifier, false);
```

If the expression does not result in a domain element null will be returned. IStructuralFeature is described in the API help.

## 2.7.4.1 InstalledOperations

Each OCL service implements a different set of OCL operations. It is possible to discover a list of installed operations for a given OCL service using the InstalledOclOperations property. This property will contain an entry for every installed operation, in addition it will contain operations with duplicate names when an operation has many overloads.

```
foreach (IOclOperation currentOperation in EcoSpace.OclPs.InstalledOclOperations)
    Console.WriteLine(currentOperation.Name);
```

## 2.7.4.2 Creating a string operation

Custom OCL operations may only be registered in the IOclService and IActionLanguageService. This is due to the fact that these are evaluated in memory, to implement an IOclPsService operation would involve a way to translate the operation to SQL, and this functionality is not currently supported in ECO. There is a class named OclOperationBase which may be used to simplify the process of creating custom OCL operations, when descending from this base class you only need to override two methods; Init and Execute.

The following example illustrates how to create an operation which reverses a string.

```
public class ReverseStringOperation : OclOperationBase
{
    //Recommended approach. Have a static method that registers the operation into
    //the correct OCL services
    //(as some may be intended for the Action Language Service only)
    public static void InstallOperation(IEcoServiceProvider serviceProvider)
    {
        if (serviceProvider == null)
            throw new ArgumentNullException("ServiceProvider");

        ReverseStringOperation impl = new ReverseStringOperation();
        serviceProvider.GetEcoService<IOclService>().InstallOperation(impl);
        serviceProvider.GetEcoService<IActionLanguageService>().InstallOperation(impl);
    }

    //Called when the OCL operation is installed
    protected override void Init()
    {
        //Define the operation name
        string operationName = "reverse";

        //Specify the source must be a string
        IOclType sourceType = Support.StringType;

        //Initialise using
        // Operation name
        // Parameters - The first is the source
    }
}
```



```

// Return type - In this case the same as the source
InternalInit(
    operationName,
    new IOclType[] { sourceType },
    sourceType);
}

//Executed when the operation needs to be processed
public override void Evaluate(IOclOperationParameters oclParameters)
{
    //Get the first value (the source) as an element, and cast it to a string
    string value = oclParameters.Values[0].Element.GetValue<string>();

    //Use a StringBuilder to reverse the string's characters
    StringBuilder resultBuilder = new StringBuilder();
    for (int index = value.Length - 1; index >= 0; index--)
        resultBuilder.Append(value[index]);

    //Create a constant IElement representing the reversed string
    IElement result =
Support.VariableFactory.CreateConstant(resultBuilder.ToString());

    //Set the result of the operation
    oclParameters.Result.SetOwnedElement(result);
}
}

```

This example would be used like so

```

//You should install operations in the EcoSpace constructor!
ReverseStringOperation.InstallOperation(EcoSpace);

string reversed =
    EcoSpace.Ocl.Evaluate("Person.allInstances->first.firstName.reverse").GetValue<string>(
);

```

### 2.7.4.3 Creating a collection operation

The following operation shows how to create an operation based on a collection, and also how to specify additional parameters. Unlike the ReverseStringOperation example the InternalInit method is called with an OclResultTypeDeduceMethod parameter for the result type instead of an IOclType. This informs the evaluator to deduce the result type from the source.

```

public class SampleCollectionOperation : OclOperationBase
{
    //Register the operation
    public static void InstallOperation(IEcoServiceProvider serviceProvider)
    {
        if (serviceProvider == null)
            throw new ArgumentNullException("ServiceProvider");

        SampleCollectionOperation impl = new SampleCollectionOperation();
        serviceProvider.GetEcoService<IOclService>().InstallOperation(impl);
        serviceProvider.GetEcoService<IActionLanguageService>().InstallOperation(impl);
    }

    //Initialise the operation details
    protected override void Init()
    {
        //The operation name
        string operationName = "sample";

        //The source must be a list
        IOclType sourceType = Support.ListType;

        //1st parameter must be an integer (number of samples)
        IOclType numberOfSamplesType = Support.IntegerType;
    }
}

```

```

//Return type is the same as the source, but allow duplicates.
//This is because the source might be a collection of strings
OclResultTypeDeduceMethod resultType = OclResultTypeDeduceMethod.SourceAsBag;

//Register
InternalInit(
    operationName,
    new IOclType[] { sourceType, numberOfSamplesType },
    resultType);
}

public override void Evaluate(IOclOperationParameters oclParameters)
{
    //Get the source as a collection
    IElementCollection source =
        oclParameters.Values[0].Element.GetAsCollection();

    //Get the number of samples to take
    int numberOfSamples = oclParameters.Values[1].Element.GetValue<int>();
    if (numberOfSamples > source.Count)
        numberOfSamples = source.Count;

    double currentIndex = 0;
    double stepSize = source.Count / (double)numberOfSamples;

    //Create an element to hold the result
    IElementCollection result =
        (IElementCollection)Support.CreateNewVariable(oclParameters.Result.OclType);

    //Add samples to the result
    while (numberOfSamples > 0)
    {
        int readIndex = (int)Math.Round(currentIndex);
        if (readIndex >= source.Count)
            readIndex = source.Count - 1;
        result.Add(source[readIndex]);
        numberOfSamples--;
        currentIndex += stepSize;
    }
    oclParameters.Result.SetOwnedElement(result);
}
}

```

## 2.8 IPersistenceService

The persistence service is responsible for mediating between the EcoSpace and the data storage. Rather than a create/retrieve/update/delete approach ECO's persistence service implements create/update/delete via a single instruction to update the data storage, ECO's internal state management will track which type of operation is appropriate based on actions performed against the local object cache (creating a new object, modifying an object, or deleting an object); freeing the developer from having to concern themselves with which type of call to make.

### Persisting changes to the data storage

The most simple way to update an object to the data storage is to call the UpdateDatabase method with a single object.

```

//Create a new person and persist it
Person person1 = new Person(EcoSpace);
EcoSpace.Persistence.UpdateDatabase(person1);

//Modify an existing person and persist the changes
person1.FirstName = "John";

```

```
EcoSpace.Persistence.UpdateDatabase(person1);
//Delete an existing person and persist the deletion
person1.AsIObject().Delete();
EcoSpace.Persistence.UpdateDatabase(person1);
```

Whenever an UpdateDatabase is performed ECO will additionally remove any undo or redo blocks held by the undo service (see page 9) which reference the object that is having its changes persisted. This is in order to prevent the consumer of the business model from making changes, persisting the changes, and then undoing those changes in the local cache; effectively making the local cache out of sync with the data storage.

Usually the application consuming the business model is solely responsible for deciding when to update the data storage, but sometimes part of the business logic dictates that an update made by the model should be persisted immediately. For example, if your business model implements a custom form of pessimistic locking

```
public bool AcquireLock(PessimisticLock lock)
{
    //01: Throw an exception if lock.LockedBy is not null
    //02: Unload the lock object to make it current

    //03: Set lock.LockedBy to the current user
    //04: Attempt to update the database
    //05: Catch any optimistic locking exception
}
```

In the preceding pseudo code it is necessary to update the data storage immediately in order to ensure the custom pessimistic lock is required. If there is an active undo block then the changes performed here will be recorded by that block, the call to UpdateDatabase will then remove the block from the undo service to prevent further use of it, which could cause a problem for the application using the model as it would rightly expect the undo block to still be present. The correct approach would therefore be

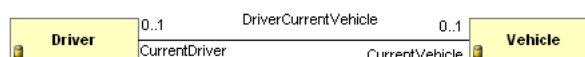
```
public bool AcquireLock(PessimisticLock lock)
{
    //01: Throw an exception if lock.LockedBy is not null
    //02: Unload the lock object to make it current

    /** Start a new undo block

    //03: Set lock.LockedBy to the current user
    //04: Attempt to update the database
    //05: Catch any optimistic locking exception

    /** In case of an exception remove the undo block
    //from the undo service
}
```

Another operation performed by ECO is to ensure that any update to the data storage is logical. If we refer back to the driver vehicle model from earlier in this document we can see that the following source code is a complete logical operation



```
Driver driver1 = new Driver(EcoSpace);
Vehicle vehicle1 = new Vehicle(EcoSpace);

EcoSpace.Persistence.UpdateDatabase(driver1);
```

But the following example is not a complete logical operation

```
Driver driver1 = new Driver(EcoSpace);
Vehicle vehicle1 = new Vehicle(EcoSpace);

//Create an association between the two new objects
driver1.CurrentVehicle = vehicle1;

EcoSpace.Persistence.UpdateDatabase(driver1);
```

This code is not a complete logical operation because there is a reference from driver1 to vehicle1 which has not yet been persisted. In order to ensure the update operation is a complete logical unit ECO will make a call to the EnsureEnclosure method on the persistence service, which ensures that all objects required to make the update are also included in the update.

In this example the enclosure occurs because the vehicle refers to an object that has not been persisted, but enclosure may also occur when a reference changes. In the Driver/Vehicle model it is likely that one of the ends of the association is marked as embedded. If the Driver end of the association is marked as embedded it means that the primary key of the Driver is embedded into the table holding the Vehicle data, so the Vehicle table would have a column named "Driver".

#### Driver

Primary key	Name
1234	John Smith

#### Vehicle

Primary key	RegistrationNumber	Driver
4321	DE 51 RED	1234

```
driver1.CurrentVehicle = null;
EcoSpace.Persistence.UpdateDatabase(driver1);
```

In the preceding data tables you can see that the driver "John Smith" is currently assigned the vehicle "DE 51 RED". In the code snippet John Smith (a.k.a. driver1) has had his vehicle unassigned. The changes made to driver1 are then persisted but in order to make the update complete "DE 51 RED" must also be updated as it is Vehicle's database table that actually holds the reference.

*Note: It is possible that neither end of the association is marked as embedded, resulting in a "link table" in the database holding the primary key of each side of the association. This is common in many-to-many associations and quite rare in one-to-one associations.*

#### Persisting a collection of objects' changes

It is also possible to persist changes to multiple instances as a single database operation. The UpdateDatabaseWithList method accepts an IObjectList parameter which may be obtained in a number of ways, the most commonly used are

### Updating all changes

```
IObjectList dirtyObjects = EcoSpace.DirtyList.AllDirtyObjects();
EcoSpace.Persistence.UpdateDatabaseWithList(dirtyObjects);
```

*Note: This code is the equivalent of EcoSpace.UpdateDatabase()*

### Updating changes captured by an undo block

```
string blockName = Guid.NewGuid().ToString();
EcoSpace.Undo.StartUndoBlock(blockName);

//Make changes here

IUndoBlock undoBlock = EcoSpace.Undo.UndoList[blockName];
IObjectList dirtyObjects = undoBlock.GetChangedObjects();
EcoSpace.Persistence.UpdateDatabaseWithList(dirtyObjects);
```

### Unloading object contents

Unloading an instance of a business class simply removes its loaded property values from the local EcoSpace cache. An object cannot be unloaded if its class has been modeled as Transient, if the EcoSpace has no persistence (which effectively makes all instances Transient), an instance can also only be unloaded if it is unmodified.

```
EcoSpace.Persistence.Unload(person1);
```

It is possible to query whether or not an object's contents have been loaded using the IsLoaded method:

```
IObjectList objects = .....;
if (!EcoSpace.Persistence.IsLoaded(objects[0]))
    .....;
```

The IObjectList in this case would most likely be the result of evaluating an OCL expression. It would be pointless obtaining the IObject reference using person1.AsObject(), IObject is merely an object-locator whereas person1 would be of the modeled "Person" type, and in order to have a reference to the modeled type the object's contents would first need to be loaded into the local EcoSpace cache.

### Efficiently retrieving lists of objects

Associations in ECO are lazy fetched, meaning that an associated object is only loaded if it is accessed.

```
Vehicle vehicle1 = {Some code to retrieve a single vehicle};

//Accessing its driver will load the driver from the data storage
Driver driver1 = vehicle1.Driver;
```

If the association is a multi-role then only the object locators are loaded when the property is first accessed, the contents of the objects are loaded when an attempt is made to access an individual object in the collection.

```
Customer customer1 = {Some code to retrieve a single customer};
```

```

//Accessing the customer's purchase orders will retrieve ID's only
if (customer1.Orders.Count > 0)
{
    //Accessing an order by index will load the single order's contents
    PurchaseOrder order1 = customer1.Orders[0];
}

```

Iterating through a list of orders using a for loop would be inefficient as it would result in a single data storage fetch per order in the collection. ECO has a number of techniques for improving the performance of fetching associated objects. The first, and most simple, is to use an enumerator to loop through the elements.

```

//Efficient approach, using an enumerator indicates your intention
//to use multiple orders in the collection. ECO will load the associated
//objects in batches of 50.
foreach (var currentOrder in customer1.Orders)
    ...

//Inefficient approach, using a specific index does not reveal any intention
//to use multiple associated orders, so ECO loads only the order at the
//specified index.
for (int i = 0; i < customer1.Orders.Count; i++)
{
    PurchaseOrder currentOrder = customer1.Orders[i];
}

```

The same applies when you evaluate an OCL expression to retrieve a collection of objects. ECO will firstly only retrieve object IDs from the data storage, it will only load the objects' contents when you attempt to access them.

```

//Only ID's are retrieved from the data storage
IObjectList list = EcoSpace.OclPs("Customer.allInstances->select(isActive)");

//The first customer in the list will have its contents loaded from the data storage
Customer customer1 = list[0].GetValue<Customer>();

```

An IObjectList is merely a collection of object locators which may be converted to instances of modeled classes. Because of this when you use an enumerator you are iterating object locators and not instances of modeled classes; as a consequence using an enumerator will not automatically fetch object contents because the step to retrieve the class instance is an additional one:

```

//Retrieve object ID's
IObjectList list = ...;

foreach (IObject locator in list)
{
    //Single operation within the loop to convert to an
    //instance of a modeled class
    Customer currentCustomer = locator.GetValue<Customer>();
}

```

If your intention is to iterate over the instances referred to by these object locators (rather than merely to use the list as the context for a second OCL evaluation) it is possible to instruct ECO to pre-load the objects in the locator list.

```

IObjectList list = ...;

//Easiest way, if you intend to iterate over all instances.

```

```
//This pre-loads the objects and returns an IList of the  
//relevant type.  
IList<Customer> customers = list.GetAsIList<Customer>();
```

Or alternatively to pre-load only a subset

```
IObjectList list = ...;  
  
//Pre-load a subset of the objects in the locator list.  
  
int firstIndex = 0;  
int lastIndex = list.Count / 2;  
EcoSpace.Persistence.EnsureRange(list, firstIndex, lastIndex);
```

### Efficiently loading related objects

The previous section shows how to efficiently load objects either in a locator list, or associated from a single object. There are circumstances where your code needs to perform a nested loop; for example to loop through every PurchaseOrder of every customer in a list.

```
//Example 1: Using OCL to eliminate the need for an outer loop  
string ocl = "Customer.allInstances->select(isActive).orders";  
var orders = EcoSpace.OclPs.Execute(ocl).GetAsIList<PurchaseOrder>();  
  
//Example 2: Pre-loading an association on a list of object locators  
IObjectList list = EcoSpace.OclPs.Execute("Customer.allInstances->select(isActive)");  
EcoSpace.Persistence.EnsureRelatedObjects(list, "Orders");
```

### Related Topics

For information regarding the GetAllWithCondition see the Version Service (see page 141).

## 2.8.1 Multi user concurrency

TODO

## 2.9 IExtentService

Each instance of an EcoSpace contains an extent for every class in the model. If a request to IExtentService.AllInstances is made, or if an OCL expression is evaluated that specifies "SomeClass.AllInstances" an object locator will be created for every instance in the data storage and held in the extent service. Any subsequent request to the extent service for the same information will return this cached collection rather than accessing the data storage again. Access is only made to the data storage again if

1. The extent for the specific class has not yet been requested for this EcoSpace instance.
2. The extent service of the current EcoSpace has previously been instructed by the programmer to unload the extent for the specified class.

Note that the extent service's state is unique per EcoSpace and not shared amongst multiple EcoSpaces. So if two EcoSpace instances request the extent for the same class they will both make a request to the data storage.

The extend manages two pieces of information:

Type	Description
AllInstances	This is a list of object locators, one for each instance of the specified type.
AllLoadedInstances	This again is a list of object locators. Instead of being one locator for every instance of the specified type it contains a list of all previously retrieved customer locators.  For example, if you were to navigate to a customer via one of its purchase orders this would result in the customer's locator being added to the AllLoadedInstances list for the Customer class.

```
//Get all previously loaded customers
int loadedCustomerCount = EcoSpace.Extents.AllLoadedInstances(typeof(Customer)).Count;

//Get a locator list for all customers - data storage access is required
IObjectList customerLocators = EcoSpace.Extents.AllInstances(typeof(Customer));

//Get a locator list for all customers - not data storage access required
IObjectList customerLocators2 = EcoSpace.Extents.AllInstances(typeof(Customer));
```

### Unloading an extent

An EcoSpace instance should be thought of as a unit of work, or as the equivalent of a database transaction. It is for this reason that the extents are cached, so that behaviour is predictable and to improve performance. There may sometimes be circumstances where you wish to invalidate the extent for a specific class.



```
//Invalidate the extent for Customer
IClass classToUnload = (IClass)EcoSpace.TypeSystem.GetClassifierByType(typeof(Customer));
EcoSpace.Extents.Unload(classToUnload);
```

If for example you are writing a WinForm application which has an ExpressionHandle with the expression "Customer.allInstances", unloading the extent for the Customer class will cause the ExpressionHandle to reevaluate and display any new instances that may have been created by other users. Unloading the extent does not unload any objects' data contents.

### Subscribing to changes

It is possible to subscribe to two events of the extent service. Using the SubscribeToObjectAdded method it is possible to register an observer which will be called back each time a new locator is added to the extent service. This could be due to creating a new instance of a class or due to loading an existing instance from the data storage.

```
//A simple class to show a message box containing the class name of
//the locator added
public class NewObjectNotifier : SubscriberAdapterBase
{
    public NewObjectNotifier(object actualSubscriber)
        : base(actualSubscriber)
    {
    }

    protected override void DoReceive(object sender, EventArgs e, object
actualSubscriber)
    {
        var args = (ElementChangedEventArgs)e;
        MessageBox.Show(args.Element.UmlType.Name);
    }
}

//Example usage
//Create the subscriber
var subscriber = new NewObjectNotifier(this);

//Subscribe to locators being added for any class, this is achieved
//using the "ECOModelRoot" class, which is the superclass of all
//classes within the model.
EcoSpace.Extents.SubscribeToObjectAdded(subscriber, "ECOModelRoot");
```

To subscribe to locators being removed you may use the SubscribeToObjectRemoved. This method is almost the mirror of SubscribeToObjectAdded. Instead of triggering whenever an object is created or loaded the subscriber is triggered whenever an object is deleted. The trigger is executed as soon as customer.AsObject().Delete() is executed, rather than when a deleted object is updated using the persistence service (which causes the object locator to be relinquished).

Note that the subscriber is not triggered when an extent is unloaded, nor when an object's contents are unloaded. Unloading an object's contents merely causes the local cached data values to be unloaded, it does not result in the EcoSpace relinquishing the locator for the object (the object's identity is still known).

## 2.10 IVersionService

Using the version service via the IVersionService interface, the developer is able to retrieve historical information about objects that have been identified as "Versioned" in the ECO model.

Object instances of classes that have been marked as Versioned are treated differently by the ECO persistence mechanism. By default each object within the database will have two additional columns, "TimeStampStart" and "TimeStampStop". These columns identify the life span of versioned objects.

Each time UpdateDatabase is executed a new integer timestamp is value allocated, and the current date/time recorded against it. These integers are used to identify at which date/time a versioned object instance is created, modified, or deleted. When a new object instance is created the current timestamp is entered into its TimeStampStart column, and 2147483647 is entered into its TimeStampStop column, this records when the object came into existence, and the high TimeStampStop indicates that this row in the database is the current "live" data for the object.

TimeStampStart	TimeStampStop	ECO_ID	FullName
10	2147483647	5	Miss Jane Smith

When a versioned object is modified the TimeStampStop column of the live row is updated to the current timestamp value, and a new row is inserted into the table. This new row has the same ECO\_ID (the unique identifier for an ECO object instance), the current timestamp for TimeStampStart, and the new modified attribute values.

TimeStampStart	TimeStampStop	ECO_ID	FullName
10	10	5	Miss Jane Smith
11	2147483647	5	Mrs Jane Jones

Finally, when a versioned object is deleted, the TimeStampStop column of the live row is updated with the current timestamp - 1.

TimeStampStart	TimeStampStop	ECO_ID	FullName
10	10	5	Miss Jane Smith
11	11	5	Mrs Jane Jones

To enable versioning on a class you must

1. Set Versioned = True on the class in the modeler.
2. Set the following properties to True on the PersistenceMapper that your EcoSpace uses
  - UseClockLog
  - UseTimestampColumn
  - UseTimestampTable

There is also a VersionGranularity property on the persistence mapper. If set to its default value time span of 00:00:00 a new version will be created for every call to UpdateDatabase. If set higher it is possible to instruct ECO to consider changes to

the same object within a specific window of time to be considered the same update, and not to create a new version of the object being updated.

### Retrieving a historical version of an object instance

To retrieve a historical version of an instance you will first need to convert a specific date and time to a version number. Once you have the correct version number it is simple to retrieve the historical version of that object, all historical object versions are read only.

```
//Specify the date and time
var pointInTime = new DateTime(...);

//Convert the date/time to a version number
int versionNumber = EcoSpace.Versioning.VersionAtTime(pointInTime);

Person person1 = {Some code to get a customer instance};
Person historical =
    EcoSpace.Versioning.GetVersion(versionNumber,
    customer1.AsIOBJECT()).GetValue<Person>();

MessageBox.Show(string.Format("Changed from {0} to {1}",
    historical.FullName, person1.FullName));
```

### Showing all changes to an object

The GetChangePointCondition method creates an instance of AbstractCondition, which may then be used with the persistence service to retrieve a full history of an individual object. Starting from an ECO WinForms application add the following code to the constructor of your form.

```
//Create a person
Person person1 = new Person(EcoSpace);

//Set the person's name and update the database
person1.FullName = "Miss Jane Smith";
EcoSpace.UpdateDatabase();

//Sleep for 1 second
Thread.Sleep(1000);

//Change the person's name and update the database
person1.FullName = "Mrs Jane Jones";
EcoSpace.UpdateDatabase();
```

Assuming you have correctly versioned the Person class and set up versioning on the EcoSpace's persistence this code will create two versions of a person when the application starts. To display this history in a WinForm DataGridView execute the following steps

1. Set rhRoot.StaticValueTypeName to "Collection(Person)" - without the quotes.
2. Add a DataGridView to your form, and use rhRoot as its data source.
3. Add the following additional code to the bottom of your form's constructor

```
var condition =
    EcoSpace.Versioning.GetChangePointCondition(
        rhRoot.Element, //Object to be retrieved
        0, //Earliest version to retrieve
```

```

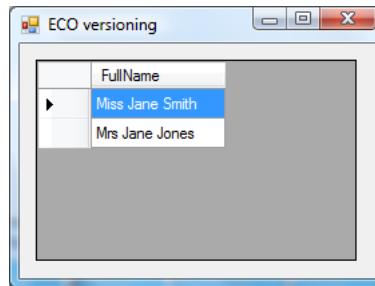
        EcoSpace.Versioning.CurrentVersion //Latest version to retrieve
    );

    //Retrieve all versions of this person
    IObjectList historicalVersions =
        EcoSpace.Persistence.GetAllWithCondition(condition);

    //Set the reference handle to hold the list of versions
    rhRoot.SetElement(historicalVersions);

```

Running the application should give you a form that looks similar to the following illustration. Note that you cannot use the XML persistence for this example.



### Adding each version's date and time

To show the date and time of each version we first need to add a code-derived column to the expression handle

1. Bring up the Columns editor on rhRoot.
2. Click the drop-down arrow to the right of the "Add" button and select "EventDerivedColumn".
3. Name the column VersionTimeStamp.
4. Set its TypeName property to System.DateTime.
5. Click OK.

Now that the code-derived column has been added add the following code to the DeriveValue event of rhRoot.

```

private void rhRoot_DeriveValue(object sender, DeriveEventArgs e)
{
    switch (e.Name)
    {
        case "VersionTimeStamp":
            //Get the current version number of the current row
            //Each row will have a different version number
            int versionNumber = EcoSpace.Versioning.ElementVersion(e.RootElement);

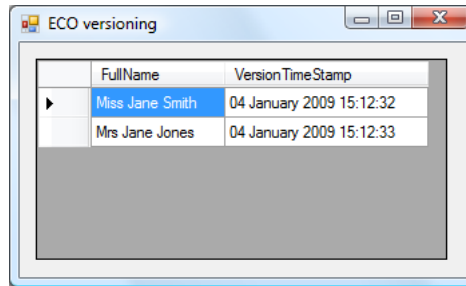
            //Convert the version number to a date/time
            DateTime timeStamp = EcoSpace.Versioning.TimeForVersion(versionNumber);

            //Set this date/time as the value to display in the data grid
            e.ResultElement = EcoSpace.VariableFactory.CreateConstant(timeStamp);
            break;

        default:
            throw new NotImplementedException(e.Name);
    }
}

```

Once you have added the new column to the data grid you should see something like the following when you run the application.



	FullName	Version Time Stamp
▶	Miss Jane Smith	04 January 2009 15:12:32
	Mrs Jane Jones	04 January 2009 15:12:33

## 2.11 ICacheContentService

## 2.12 Subscriptions

When creating a business model it is often necessary to create members that have no persistent value but are instead calculated from other values. ECO supports this feature via "Derived" members. When creating a model it is possible to mark a property / association as Derived, indicating to ECO that the value needs to be calculated. When an attempt is made to read the value of a derived member ECO will perform the necessary actions to calculate its value. The calculated value is then stored away in the local cache, so that what it is read again no further calculations are required.

If calculating the value of the derived member is costly then storing the result in the cache will obviously save resources whenever the value is read again. However, if the calculated value were just to be stored away indefinitely it could easily become "stale." For example if Person.FullName were to be derived using the following OCL expression

```
salutation + ' ' + givenName + ' ' + familyName
```

If the value of any of these three members changes then rereading a stale cached value would result in an incorrect result. This is where the ECO subscription mechanism comes in.

In the OCL derived member example above as ECO parses the OCL expression above in order to calculate the result it will need to access various members of the model; in this case Person.Salutation, Person.GivenName, and Person.FamilyName. Each derived member has its own "subscriber", as the ECO OCL evaluator accesses the value of a member it adds this subscriber to a list of parties interested in knowing when the member's value changes.

Once the subscriptions have been placed with the relevant members and the result determined the derived member's value will be cached. Subsequent reads of the derived member will return the cached value. When one of the three members in the expression change they will notify every subscriber that has been registered with them. In this case the only subscriber will be the one owned by FullName, when this subscriber's Receive() method is executed it will invalidate its cached value.

Any subsequent attempt to read the value of the derived member will see that there is no cached value for it and cause the OCL evaluator to reprocess the expression and replace any required subscriptions.

### Subscribing to derived members

A derived member may use any type of member as part of its calculated value. Associations, persistent members, transient members, and also other derived members. Take a simple class as an example, consisting of only three members.

Name	Type	Derivation code
Transient1	Transient	
Derived1	Code derived	<pre>private string Derived1Derive() {     System.Diagnostics.Debug.WriteLine(" Model: Calculating Derived1");     return "D1 + " + Transient1; }</pre>
Derived2	Code derived	<pre>private string Derived2Derive() {     System.Diagnostics.Debug.WriteLine(" Model: Calculating Derived2");     return "D2 + " + Derived1; }</pre>

now consider the following application code

```
private void ReadDerived1(Class_1 instance)
{
    System.Diagnostics.Debug.WriteLine("App: Reading derived1");
    System.Diagnostics.Debug.WriteLine(" Result = " + instance.Derived1);
}

private void ReadDerived2(Class_1 instance)
{
    System.Diagnostics.Debug.WriteLine("App: Reading derived2");
    System.Diagnostics.Debug.WriteLine(" Result = " + instance.Derived2);
}

private void ChangeTransient(Class_1 instance, string value)
{
    System.Diagnostics.Debug.WriteLine("App: Changing transient1 to " + value);
    instance.Transient1 = value;
}
```

These instructions have only been made into methods in order to log how the application is using the domain object, and to make the steps easier to demonstrate.

Step	Instruction	Output
1	ChangeTransient(instance, "Hello world");	App: Changing transient1 to Hello world
2	ReadDerived1(instance);	App: Reading derived1 Model: Calculating Derived1 Result = D1 + Hello world

3	ReadDerived2(instance);	App: Reading derived2 Model: Calculating Derived2 Result = D2 + D1 + Hello world
4	ChangeTransient(instance, "Goodbye world");	App: Changing transient1 to Goodbye world
5	ReadDerived2(instance);	App: Reading derived2 Model: Calculating Derived2 Model: Calculating Derived1 Result = D2 + D1 + Goodbye world

1. The transient member has its value changed. This is just to start with a meaningful value. It has no effect on derived members as none of them have been accessed yet and therefore have not placed any subscriptions.
2. The value of Derived1 is read. This calculation is based only on Transient1. The value is calculated and stored in the local cache. Derived1 places a subscription on Transient1.
3. The value of Derived2 is read. This calculation is based only on Derived1. When the value of Derived1 is read ECO sees that it has previously been calculated and cached, the calculation is not performed again, instead the cached value is returned. Derived2 places a subscription on Derived1.
4. The transient member is modified. As a consequence a change notification is sent to all of its subscribers.
  1. Derived1 receives a notification that one of the members it has subscribed to has been modified.
  2. Derived1 invalidates its cached value.
  3. Derived1 notifies all of its subscribers that its value has possibly changed.
  4. Derived2 receives a notification that one of the members it has subscribed to has been modified.
  5. Derived2 invalidates its cached value.
5. The value of Derived2 is read.
  1. There is no cached value for Derived2 so its value is recalculated.
  2. Derived2 reads the value of Derived1
  3. There is no cached value for Derived1 so its value is recalculated.
  4. Derived1's value is cached.
  5. Derived2's value is cached.

### Auto subscription

As mentioned previously the OCL evaluator will automatically subscribe to any members it accesses during evaluation of an expression. In the previous example however the values were accessed via source code and not an evaluator, so how were the subscriptions placed?

All member values are stored in a local cache. Whenever a read/write is performed on a .NET instance of a modeled ECO class the property uses the local ECO cache to read/write the value. This means that ECO is fully aware of any time a value is touched, and as a result is able to identify which elements make up a derived member. When an attempt is made to access a derived member ECO performs the following steps

1. If there is a cached value.
  1. Return the cached value.
  2. Finish.
2. If the member has an OCL expression.

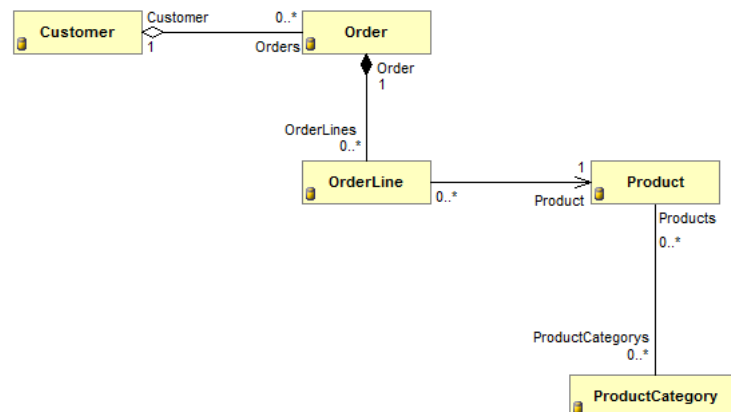
1. Evaluate the expression.
  2. Place subscriptions on accessed elements.
  3. Store the result in the cache.
  4. Return the cached value.
  5. Finish.
3. Find a method named <MemberName>Derive.
    1. The members subscriber is pushed onto the IAutoSubscriptionService's stack.
    2. The <MemberName>Derive method is executed.
    3. Any access to an element in the cache checks the ActiveSubscriber in the auto subscription service, and registers it as a party interested in being notified when the element's value changes.
    4. The <MemberName>Derive method returns a result.
    5. The member's subscriber is removed from the auto subscription service's stack.
    6. Cache the result.
    7. Finish.

## 2.13 ITypeSystemService

The type system services allows your application to inspect your model in great detail at runtime.

### Short example - Identifying all classes used by an EcoSpace

The first and most simple example shows how to identify all classes used by the EcoSpace. If the EcoSpace uses multiple class packages this list will include all classes of all packages used.



```

foreach (IClass c in ecoSpace.TypeSystem.AllClasses)
    Trace.WriteLine(c.Name);
  
```

The output from the preceding source code (viewed in the Debug->Windows->Output window in Visual Studio) is as follows

```

ECOModelRoot
Customer
Order
  
```



```

OrderLine
Product
ProductCategory
ProductCategoryProducts

```

In the output you will see five class names you would expect to see after looking at the UML diagram for the model but there are two additional names you may not have expected, the first and last in the list. When ECO builds a runtime representation of your model it inspects it for a common superclass from which all classes ultimately descend, if no such class exists ECO will add an ECOModelRoot class which acts as an equivalent of System.Object for your model.

The other unexpected class was ProductCategoryProducts. When you model a many to many association (in this case between Product and ProductCategory) ECO creates an implicit association class based on the name given to the association. You wont see a business class source file generated for this class, it is actually created as an embedded class of the package, this class exists only to provide additional meta-information to ECO at runtime and for creating database structures. Of course if you explicitly define an association class between Product and ProductCategory you will get a full business class generated in which you may define additional members.

Changing the source code as follows

```

foreach (IClass c in ecoSpace.TypeSystem.AllClasses)
    Trace.WriteLine(
        string.Format("Name={0} Implicit={1} IsLink={2}",
            c.Name, c.IsImplicit, c.IsLinkClass)
    );

```

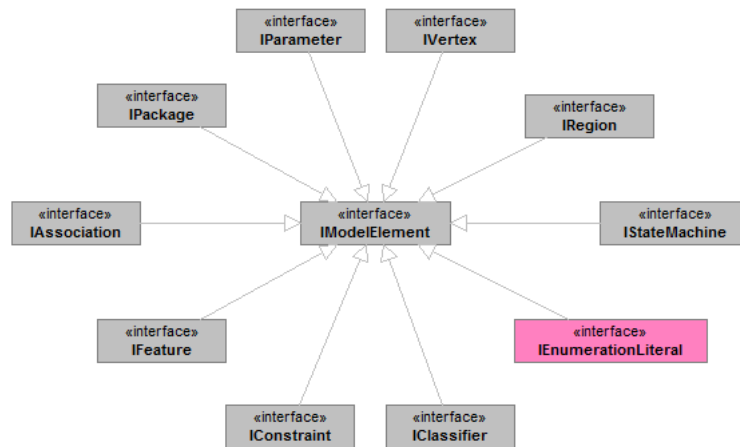
Will provide the following data, which I have formatted as a table.

Name	Implicit	IsLink
<b>ECOModelRoot</b>	<b>True</b>	<b>False</b>
Customer	False	False
Order	False	False
OrderLine	False	False
Product	False	False
ProductCategory	False	False
<b>ProductCategoryProducts</b>	<b>True</b>	<b>True</b>

In the following UML diagrams I have used the following colour scheme.

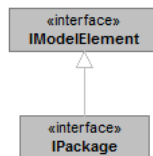
1. Grey : Elements which are created once when the details of the runtime model is first established.
2. Purple : Elements which hold meta information about values which may be created at runtime, such as variables and constants created by the IVariableFactoryService (see page 18).

## 2.13.1 IModelElement



The IModelElement interface is the base interface for almost all ECO model meta information. This interface allows you to determine the Name of the element, which Package it belongs to, and a collection of OCL constraints. The modeler also permits IModelElement descendants such as classes and their members to have tagged values (see page 151) assigned to them which may be read at runtime.

## 2.13.2 IPackage



A list of packages used by an EcoSpace may be obtained using from the `EcoSpace.TypeSystem.AllPackages` property. Each package contains a collection of classes and associations that were modeled within it.

```

foreach (IPackage package in ecoSpace.TypeSystem.AllPackages)
{
    Debug.WriteLine(
        string.Format("ID={0} Classes={1} Associations={2}",
            package.Id, package.Classes.Count, package.Associations.Count)
    );
}
  
```

The output from the preceding code would be something like

```

ID={SomeGUID} Classes=5 Associations=4
  
```

Based on the simple Customer/Order model defined at the start of this section you will see there are five explicitly modeled classes and four associations. If you think back to the early source code example there were two additional classes `ECOModelRoot` and `ProductCategoryProducts`, these do not appear within the `Classes` collection because they were not explicitly modeled. Adding the following code within the above loop

```

foreach (IPackage package in ecoSpace.TypeSystem.AllPackages)
{
    //Previous code omitted
    foreach (IModelElement element in package.OwnedElements)
        Debug.WriteLine(
            string.Format(" Element={0} Type={1}", element.Name, element.GetType().Name)
        );
}

```

Will result in the following output

```

ID={SomeGUID} Classes=5 Associations=4
Element=Order Type=UmlClass
Element=ProductCategory Type=UmlClass
Element=OrderLine Type=UmlClass
Element=Product Type=UmlClass
Element=Customer Type=UmlClass
Element=OrderCustomer Type=UmlAssociation
Element=OrderLineOrder Type=UmlAssociation
Element=OrderLineProduct Type=UmlAssociation
Element=ProductCategoryProducts Type=UmlAssociation

```

Note how ProductCategoryProducts is merely an association. This is because the information within an IPackage reflects how exactly how you modeled it. When ECO initialises its runtime model it is necessary to implicitly create items in order to make the model execute. For example I mentioned earlier how an ECOModelRoot class is created if there isn't a single common super class for all classes in a package, if the EcoSpace were to consume two modeled packages which had no dependencies upon each other then there couldn't possibly be a common super class. In this case ECO would certainly need to create an implicit super class (akin to a persistent System.Object), but which IPackage would this super class belong to? The answer is that it wouldn't belong to either. The IPackages' meta-information remains unaltered, it is the EcoSpace's run meta-information that hold this implicit class, along with implicit classes for associations.

The previous output lists five classes and four associations. The output from the following source code (which uses the EcoSpace's TypeSystem)

```

foreach (IClass c in EcoSpace.TypeSystem.AllClasses)
    Debug.WriteLine("Class=" + c.Name);

```

will show seven classes

```

Class=ECOModelRoot
Class=Customer
Class=Order
Class=OrderLine
Class=Product
Class=ProductCategory
Class=ProductCategoryProducts

```

ECOModelRoot was added to the EcoSpace's runtime TypeSystem to cater for not having a common super class. In addition to having an association named ProductCategoryProducts there is also an implicit association class created. The reason this association class needs to exist is quite obvious. When changes are made to objects' state within an EcoSpace ECO identifies which instances' changes need to be persisted to the datastore by identifying each modified instance as "Dirty" (modified). When it comes to modifying associations in a one-to-one or a one-to-many association the ID of the linked

instance is stored in a single end of the association (e.g. the Order identity is embedded into the OrderLine). So adding an OrderLine to an Order will mark the OrderLine dirty and not the order, so ECO knows that only the OrderLine needs to be persisted to the datastore.

There are two scenarios however where neither end of the association is considered dirty. In a many-to-many association neither side of the association is considered dirty because a database table can typically only hold single values, so neither side's database table can hold a collection of identities. In such a situation it is common practise when writing a database application to create a link table, consisting of two IDs (one for each side of the association). By creating the implicit class for a many-to-many association ECO is doing the same thing, not only does it identifying the fact that a link table is required within the database but it also enables ECO to identify which parts of the association are dirty (A1--B1 was removed, A1--B2 was added). When modifying a many-to-many association only instances of this link-class are considered dirty.

The other scenario is the case where neither side of an association is embedded. This means that the identity of neither side of the association is stored in the opposite side. In this case ECO will again create an implicit link class for storing the association.

---

## 2.13.3 ITaggedValue

ITaggedValue is a name/value pair of two strings. Anything implementing IModelElement has a collection of tagged values which may be obtained via its TaggedValues property. Elements such as classes, properties, and methods may be decorated with named values using the modeler during design time; these tagged values may then be read at runtime by the application.

For example if you select a class (Class1) in the model, click the TaggedValues editor, and then add a tagged value with the name "MyCompany.DisplayName" the tagged value could be read as follows

```
IClass c = EcoSpace.TypeSystem.GetClassByType( typeof( Class1 ) );
ITaggedValue tv = c.GetItemByTag( "MyCompany.DisplayName" );
SomeLabel.Text = tv.Value;
```

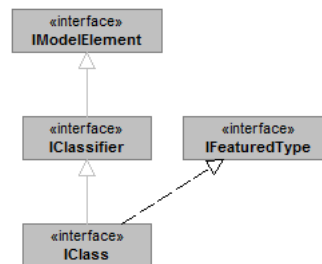
---

## 2.13.4 IStructuralFeature

A structural feature is anything on a class which holds state information. A method on a class holds no state and is therefore not a structural feature. Any modeled element on a class which produces a property in the generated code is considered a structural feature, this could be either an IAttribute or an IAssociationEnd.

This interface provides information about the state holder such as whether it is persistent / derived / transient, if it is derived / derived settable, and the IClassifier which represents the .NET type of the member. The members of IStructuralFeature , IAttribute, and IAssociationEnd are quite straight forward so wont be covered in this document, for more information about these interfaces please read the API documentation.

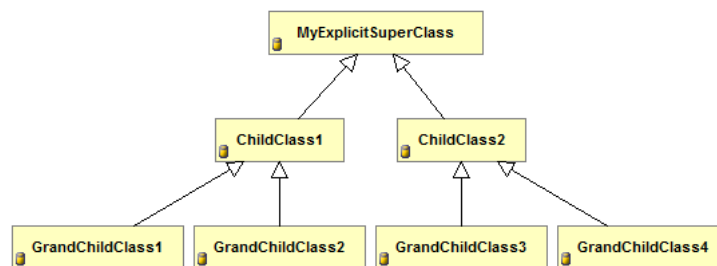
## 2.13.5 IClass



The IClass interface holds meta-information about classes. As an IModelElement it is possible to identify all tagged values assigned to it during modeling.

### Hierarchy

IClass has properties named SuperTypes and SubTypes. Although in .NET you can only descend your class from a single class the decision was made to make SuperTypes multiple in order to conform to the UML specification, SuperTypes will always contain zero or one entries. SubTypes on the other hand may obviously contain any number of entries.



Given the previous model it is possible to map the structure at runtime using the following recursive source code.

```

public void OutputEcoSpaceHierarchy()
{
    //Only if we have at least one class
    if (EcoSpace.TypeSystem.AllClasses.Count > 0)
        //Output the super class
        OutputClassHierarchy(0, EcoSpace.TypeSystem.AllClasses[0]);
}

private void OutputClassHierarchy(int indent, IClass currentClass)
{
    string indentText = new string(' ', indent);
    Debug.WriteLine(indentText + currentClass.Name);
    //Output each sub type
    foreach (IClass childClass in currentClass.SubTypes)
        OutputClassHierarchy(indent + 2, childClass);
}

```

This code would result in the following output

```

MyExplicitSuperClass
  ChildClass1

```

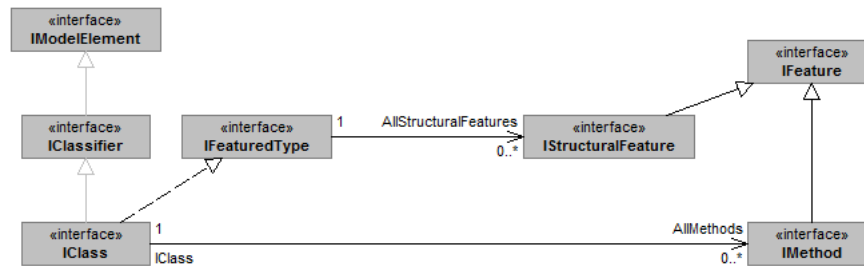
```

GrandChildClass1
GrandChildClass2
ChildClass2
GrandChildClass3
GrandChildClass4

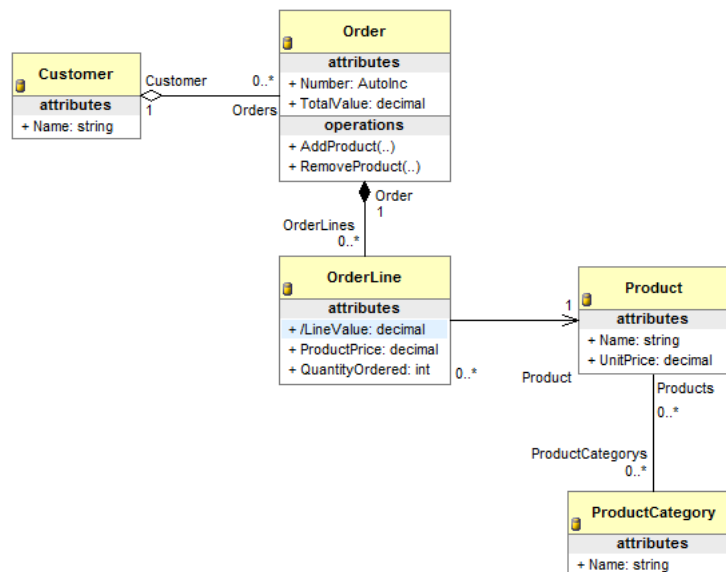
```

The AllClasses property is always sorted into a hierarchical order. A sub type will never have a lower index in the collection than its super type, ultimately the IClass at index zero will always be the root class for the entire runtime model, whether it is implicitly created or explicitly created as in this example.

### Model hierarchy



In addition to being an IModelElement the IClass interface also descends from IFeaturedType. This type is used for modeled classes and for adhoc query classes when executing a query in OCL or LINQ which returns a tuple (collection of data rather than instances of modeled types).



Using the IFeaturedType.AllStructuralFeatures property and the IClass.AllMethods properties it is possible to determine the structure of a class.

```

public void ShowOrderClassStructure()
{
    OutputClassStructure(EcoSpace.TypeSystem.AllClasses.GetItemByName("Order"));
}

private void OutputClassStructure(IClass currentClass)
{
    foreach (IFeature feature in currentClass.AllStructuralFeatures)

```

```

{
    Debug.WriteLine(
        string.Format("Feature - {0} {1} {2}",
            feature.Visibility,
            feature.Name,
            feature.FeatureType)
    );
}
foreach (IMethod method in currentClass.AllMethods)
{
    Debug.WriteLine(
        string.Format("{0} {1}",
            method.Visibility,
            method.Name)
    );
}
}

```

## Output

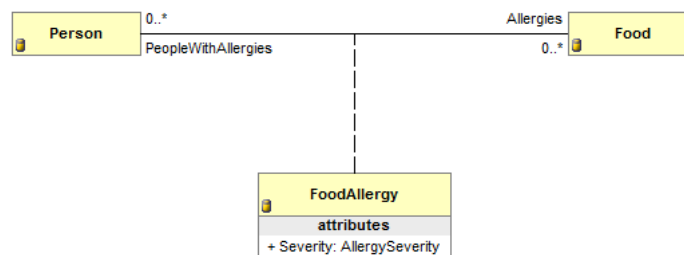
```

Feature - Public_ Customer AssociationEnd
Feature - Public_ OrderLines AssociationEnd
Feature - Public_ Number Attribute
Feature - Public_ TotalValue Attribute
Method - Public_ AddProduct
Method - Public_ RemoveProduct

```

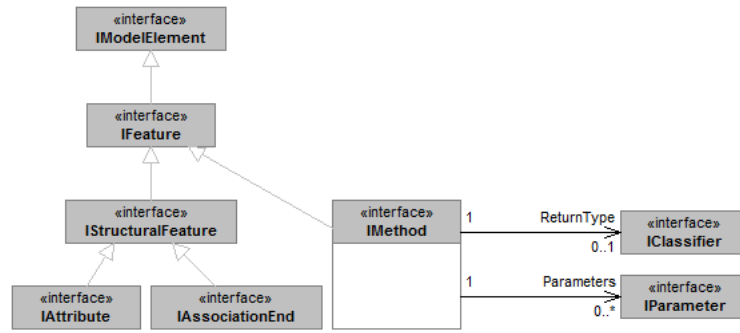
## Association classes

If the class reference is an association class, either implicit or explicit, its Association property will identify the association it represents.



With a reference to the FoodAllergy's IClass it is possible to obtain the IAssociation reference it represents and obtain additional information about the association; such as whether the association is derived / transient / persistent, or to obtain information about the classes at either end of the association (Person / Food) and the multiplicity of the association ends.

**Class features**



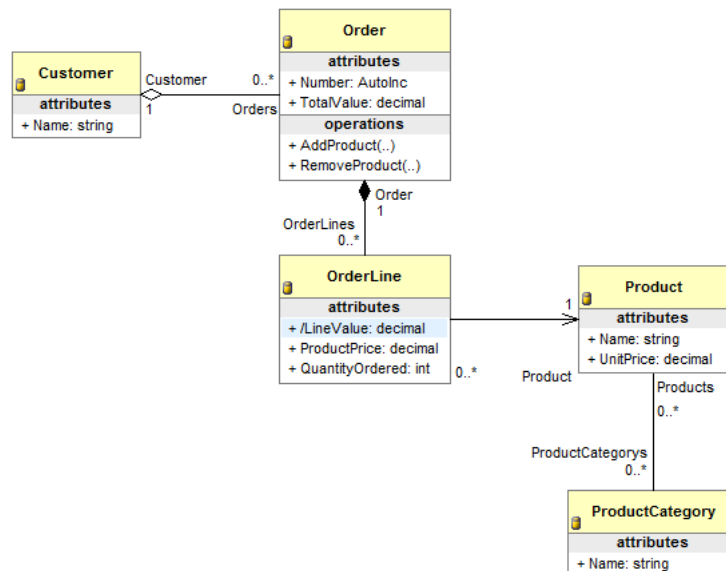
When accessing `IClass.AllStructuralFeatures` the result will contain structural features modeled in the class and all structural features it inherits from its ancestor classes. Iterating this collection of `IStructuralFeature` will give every structural feature available to the class, if you wish to iterate only structural features introduced by the current you can use the `FirstOwnStructuralFeatureIndex` property.

```

IClass c = EcoSpace.TypeSystem.GetClassByType(typeof(BaseClass));
for (int i = c.FirstOwnStructuralFeatureIndex; i < c.AllStructuralFeatures.Count; i++)
{
    IStructuralFeature f = c.AllStructuralFeatures[i];
    Console.WriteLine(f.Name);
}
    
```

**Class meta-data example**

This example uses the previous Customer/Order model.



```

public Form1(EcoProject20.EcoProject20EcoSpace ecoSpace)
{
    
```



```
//Code omitted
IClass c = EcoSpace.TypeSystem.GetClassByType(typeof(Order));
OutputClassInfo(c);
}
```

A class is considered to be either Persistent or Transient, `c.Persistent` reflects the persistence state of the class. The `DefaultStringRepresentation` is the expression that is evaluated whenever the "asString" OCL expression is evaluated against an object instance.

```
private void OutputClassInfo(IClass c)
{
    Console.WriteLine("Name : {0}", c.Name);
    Console.WriteLine(" Persistent : {0}", c.Persistent);
    Console.WriteLine(" DefaultStringRepresentation : {0}",
c.DefaultStringRepresentation);

    Console.WriteLine(" Methods");
    OutputClassMethods(c);

    Console.WriteLine(" Attributes");
    OutputClassAttributes(c);

    Console.WriteLine(" Association ends");
    OutputClassAssociationEnds(c);
}
```

```
[Output]
Name : Order
 Persistent : True
 DefaultStringRepresentation : self.Number
```

`IClass.AllMethods` contains method information for each modeled method on the class. This will contain inherit methods, but not methods which were not added via the modeler.

```
private void OutputClassMethods(IClass c)
{
    foreach (IMethod m in c.AllMethods)
    {
        Console.WriteLine(" Name : {0}", m.Name);
        if (m.ReturnType != null)
            Console.WriteLine(" Returns : {0}", m.ReturnType.Name);
        else
            Console.WriteLine(" Returns : void");
        OutputMethodParameters(m);
    }
}

private void OutputMethodParameters(IMethod m)
{
    foreach (IParameter p in m.Parameters)
        Console.WriteLine(" Parameter {0} of type {1} - direction {2}", m.Name,
p.Type.Name, p.Kind);
}
```

```
[Output]
Methods
Name : AddProduct
Returns : void
Parameter AddProduct of type Product - direction In
```

```
Name : RemoveProduct
Returns : void
Parameter RemoveProduct of type Product - direction In
```

As with AllMethods only members added to the class via the modeler will appear in AllStructuralFeatures. The list will contain both inherited and introduced members. The output of the following code is the result of inspecting the OrderLine class rather than the Order class, because the OrderLine class has a derived member. The AllStructuralFeatures list will contain both IAttributes and IAssociationEnds, the example code filters the list to show only IAttributes using the LINQ "OfType" filter.

```
private void OutputClassAttributes(IClass c)
{
    foreach (IAttribute a in c.AllStructuralFeatures.OfType<IAttribute>())
    {
        Console.WriteLine("    Name : {0} of type {1}", a.Name, a.Type_.Name);
        Console.WriteLine("    Persistent : {0}", a.Persistent);
        if (a.IsDerived)
        {
            Console.WriteLine("        Derived and settable : {0}", a.IsReverseDerived);
            if (a.DeriveAndSubscribeMethod != null)
                Console.WriteLine("        Derived using method : {0}",
a.DeriveAndSubscribeMethod.Name);
            else
                Console.WriteLine("        Derived using expression : {0}",
a.TaggedValues.GetItemByTag("Eco.DerivationOCL").Value);
        }
    }
}
```

```
[Output]
Attributes
Name : ProductPrice of type System.Decimal
    Persistent : True

Name : LineValue of type System.Decimal
    Persistent : False
    Derived and settable : False
    Derived using expression : quantityOrdered * productPrice

Name : QuantityOrdered of type System.Int32
    Persistent : True
```

The following code outputs meta information about IAssociationEnds on the Order class. As with the previous IAttribute example the list is filtered using LINQ.

```
private void OutputClassAssociationEnds(IClass c)
{
    foreach (IAssociationEnd a in c.AllStructuralFeatures.OfType<IAssociationEnd>())
    {
        Console.WriteLine("    Name : {0}", a.Name);
        Console.WriteLine("    Persistent : {0}", a.Persistent);
        if (a.IsDerived)
        {
            Console.WriteLine("        Derived and settable : {0}", a.IsReverseDerived);
            if (a.DeriveAndSubscribeMethod != null)
                Console.WriteLine("        Derived using method : {0}",
a.DeriveAndSubscribeMethod.Name);
            else
                Console.WriteLine("        Derived using expression : {0}",
a.TaggedValues.GetItemByTag("Eco.DerivationOCL").Value);
        }
    }
}
```

```
        Console.WriteLine("        Navigable : {0}", a.IsNavigable);
        Console.WriteLine("        Multiplicity : {0}..{1}", a.Multiplicity.Lower,
a.Multiplicity.Upper);
    }
}
```

```
[Output]
Association ends
Name : Customer
    Persistent : True
    Navigable : True
    Multiplicity : 1..1
Name : Lines
    Persistent : True
    Navigable : True
    Multiplicity : 0..2147483647
```

If the association were derived the output would reveal how it is derived, either as an OCL expression or via a method call. In addition it would also indicate whether the member is derived **and** settable (IsReverseDerived). This is also available on IAttribute as it is inherited from IStructuralFeature.

# 3 Registering custom services

# 4 Replacing standard ECO services

## 4.1 Replacing the ExternalIdService

## 4.2 A validating IPersistenceService

Constraints in ECO are not enforced by default. This is because it is up to the application developer to decide when to evaluate constraints and also how to handle constraints when they are broken. One approach is to prevent the user from saving changes when there are broken constraints, however a sensible backup strategy is to ensure there are no broken constraints when the datastorage is updated; this protects the persistent data from becoming corrupted if the application developer neglects to enforce constraints at a single point in the application.

The easiest way to replace the IPersistenceService for an EcoSpace is to descend a new class from the ChainedPersistenceServiceBase class. In this following code sample you will see the following

1. A constructor is added accepting the IEcoServiceProvider. This is done so that references to other services may be obtained where necessary.
2. The NextPersistenceService property is set. This ensures that all persistence requests are passed on to the real implementer.
3. UpdateDatabaseWithList<T> is overridden. This is so that the objects being updated may be validated first.

This method uses the DroopyEyes.EcoExtensions.Validation.ModeledConstraintProvider class to obtain constraints for a given instance, this class is used to simplify the example.

```
public class ValidatingPersistenceService : ChainedPersistenceServiceBase
{
    readonly IEcoServiceProvider ServiceProvider;

    public ValidatingPersistenceService(IEcoServiceProvider serviceProvider)
        : base()
    {
        if (serviceProvider == null)
            throw new ArgumentNullException("ServiceProvider");

        //Save the service provider reference, and set NextPersistenceService
        ServiceProvider = serviceProvider;
        NextPersistenceService = ServiceProvider.GetEcoService<IPersistenceService>();
    }

    public override void UpdateDatabaseWithList<T>(IEnumerable<T> list)
    {
        var constraintProvider = new ModeledConstraintProvider();

        foreach (IObject instance in list)
        {
            //Ignore deleted objects
        }
    }
}
```

```

        if (instance.Deleted)
            continue;

        //Get a list of IConstraint instances for the object being updated
        var constraints = new List<IConstraint>();
        constraintProvider.GetConstraintsForObject(instance, constraints);

        //Find the first broken constraint
        var brokenConstraint = constraints.FirstOrDefault(c => !c.IsValid);

        //If a constraint is broken throw an exception showing the
        //object class + the constraint name
        if (brokenConstraint != null)
            throw new Exception(instance.AsObject.GetType().Name + ":" +
                brokenConstraint.Name);
        base.UpdateDatabaseWithList<T>(list);
    }
}

```

To install the service register it when the EcoSpace becomes active by overriding the Active property.

```

public override bool Active
{
    get
    {
        return base.Active;
    }
    set
    {
        base.Active = value;
        if (Active)
            RegisterEcoService<IPersistenceService>(new
                ValidatingPersistenceService(this));
    }
}

```

When an attempt is made to call UpdateDatabase on an EcoSpace where one of the objects has a broken constraint an exception will be thrown with a message similar to

Person : FullName required

## Index

-

- 28, 49

\*

\* 29

/

/ 29

+

+ 28, 50

&lt;

&lt; 29

&lt;= 30

&lt;&gt; 30

=

= 30

&gt;

&gt; 31

&gt;= 31

## A

A validating IPersistenceService 160

Abs 85

Acos 85

Add 129

AddDays 66

AddHours 66

AddMilliseconds 67

AddMinutes 67

AddMonths 67

AddSeconds 68

AddTicks 68

AddYears 68

AllInstances 31

AllInstancesAtTime 50

AllLoadedObjects 51

AllSubClasses 51

AllSuperClasses 51

Append 54

AsBag 54

AsCommaList 54

Asin 85

AsSeparatedList 55

AsSequence 56

AsSet 56

AssociationEnds 52

AsString 52

At 52

Atan 86

Atan2 86

AtTime 53

Attributes 53

Average 32

## B

BigMul 86

## C

Ceiling 87

Chars 101

Clear 128

ClrSubstring 102

Collect 57

Collection operations 53

Compare 61

Concat 102

Constraints 62

Contains 103

Cos 87

Cosh 87

Count 57

Create 63, 129

Creating a collection operation 132

Creating a string operation 131

**D**

Date 69  
Date and time operations 66  
Day 69  
DayOfYear 70  
Days 69  
DaysInMonth 70  
Delete 129  
Difference 32  
Div 32  
Duration 71

**E**

EmptyList 83  
EndsWith 103  
Excluding 58  
Existing 83  
Exists 33  
Exp 87  
ExternalId 84

**F**

FilterOnType 58  
First 58  
Floor 88  
ForAll 33  
Format 104  
FormatDateTime 71  
FromBinary 71  
FromDays 72  
FromFileTime 72  
FromFileTimeUtc 72  
FromHours 73  
FromMilliseconds 73  
FromMinutes 73  
FromSeconds 74  
FromTicks 74

**G**

GetNumericValue 104

**H**

Hour 74  
Hours 74

**I**

IActionLanguageService 128  
ICacheContentService 144  
IClass 152  
IDirtyListService 8  
IEcoServiceProvider 1  
IExtentService 139  
IExternalIdService 4  
If 84  
IModelElement 149  
Implies 34  
Includes 34  
IncludesAll 59  
Including 59  
InDateRange 75  
IndexOf 59, 104  
Insert 106  
InstalledOperations 131  
Intersection 34  
InTimeRange 75  
IObjectFactoryService 16  
IOclPsService 25  
IOclService 44  
IPackage 149  
IPersistenceService 133  
IsControl 107  
IsDaylightSavingTime 75  
IsDigit 107  
IsEmpty 35  
IsHighSurrogate 108  
IsInfinity 88  
IsLeapYear 76  
IsLetter 108  
IsLetterOrDigit 109  
IsLower 110  
IsLowSurrogate 110  
IsNaN 88



IsNegativeInfinity 89  
 IsNormalized 111  
 IsNull 35  
 IsNullOrEmpty 111  
 IsNumber 111  
 IsPositiveInfinity 89  
 IsPunctuation 112  
 IsSeparator 113  
 IsSurrogate 113  
 IsSurrogatePair 114  
 IsSymbol 115  
 IStateService 6  
 IStructuralFeature 151  
 IsUpper 115  
 IsWhiteSpace 116  
 ITaggedValue 151  
 ITypeService 130  
 ITypeSystemService 147  
 IUndoService 9  
 IVariableFactoryService 18  
 IVersionService 141

## L

Last 60  
 LastIndexOf 117  
 Length 36  
 Let 84  
 Log 89  
 Log10 90

## M

Mathematical operations 85  
 Max 90  
 MaxLength 94  
 MaxValue 36  
 Millisecond 76  
 Milliseconds 76  
 Min 90  
 Minute 76  
 Minutes 77  
 MinValue 36  
 Mod 37

ModifiedSinceTimeStamp 94  
 Month 77  
 Multi user concurrency 139

## N

Negate 77, 91  
 NewGuid 95  
 Normalize 118  
 Not 37  
 NotEmpty 38  
 Now 77

## O

ObjectFromExternalId 95  
 ObjectTimeStamp 95  
 OCL operations supported by IOclPsService 28  
 OCL operations supported by IOclService 48  
 OclAsType 96  
 OclGetStates 98  
 OclGetTriggers 100  
 OclIsInState 100  
 OclIsKindOf 96  
 OclIsTypeOf 97  
 Operations support by IActionLanguageService 128  
 OrderBy 38  
 OrderDescending 39  
 OrderGeneric 39  
 Overview 1

## P

Pad 118  
 PadLeft 118  
 PadRight 119  
 Parse 97  
 PostPad 120  
 Pow 91  
 Prepend 60

## Q

Query services 22

**R**

RegExpMatch 120  
Registering custom services 159  
Reject 40  
Remainder 91  
Remove 120, 129  
RemoveAt 130  
Replace 121  
Replacing standard ECO services 160  
Replacing the ExternalIdService 160  
Round 92

**S**

SafeCast 98  
Second 78  
Seconds 78  
Select 40  
Sin 92  
Sinh 92  
Size 40  
SqlLike 41  
SqlLikeCaseInsensitive 41  
Sqrt 93  
Standard ECO services 3  
StartsWith 121  
StartTransaction, RollbackTransaction, and CommitTransaction 9  
State machine operations 98  
String operations 101  
StrToDate 121  
StrToDateTime 122  
StrToInt 122  
StrToTime 122  
Subscriptions 144  
SubSequence 60  
SubString 122  
Sum 42  
SumTime 78  
SuperTypes 124  
SymmetricDifference 61

**T**

TaggedValue 124  
Tan 93  
Tanh 93  
Terminology 2  
Ticks 78  
Time 79  
TimeOfDay 79  
TimeSpan operations 124  
TimeStampToTime 79  
TimeToTimeStamp 79  
ToBinary 80  
ToByte 125  
Today 80  
ToDouble 125  
ToFileTime 80  
ToFileTimeUtc 80  
ToInt16 125  
ToInt32 125  
ToInt64 126  
ToLocalTime 80  
ToLongDateString 81  
ToLongTimeString 81  
ToLower (Char) 123  
ToLower (String) 42  
ToLowerInvariant 123  
ToSByte 126  
ToShortDateString 81  
ToShortTimeString 81  
ToSingle 126  
TotalDays 81  
TotalHours 82  
TotalMilliseconds 82  
TotalMinutes 82  
TotalSeconds 82  
ToUInt16 126  
ToUInt32 127  
ToUInt64 127  
ToUniversalTime 82  
ToUpper (Char) 123  
ToUpper(String) 43

ToUpperInvariant 123

Trim 124

Truncate 94

TypeName 127

## U

Undo blocks 10

Union 43

UtcNow 83

## W

Working with an undo block 15

Working with the undo service 12

## X

Xor 127

## Y

Year 83