

## Context

A Canonical Object Extension is Unity Container Extension developed by Unity Contributions. This extension enables at runtime to configure and resolve a canonical object derived from collection of domain specific objects. In this context, a canonical object is a single representative object derived from closely related domain specific objects.

## Problem

- How to configure, resolve or construct a canonical object derived from collection of objects in Unity container?
- How to separate domain specific object functionality, and enable canonical or single representative object view of enterprise entity such as Customer?

## Forces

The following forces act on this extension within the context above:

- You need to maintain set of functionality in each domain specific solution, and enable the configurable canonical object view of discreet objects in Unity container.
- For example, Commercial-Off-the-Shelf (COTS) software or component which offers a certain standard functionality and allow object extensions or modifications to the standard COTS canonical object model.
- When an enterprise has developed component base software solutions and requires the single representative view of enterprise domain objects in Unity container. For example, it's often common to see enterprise objects or entities to be divided into domain specific solutions such as CRM Customer and Billing Customer.

## Benefits

Enable to maintain distinct domain specific types/objects and they functionality separately, and resolve or construct a canonical object view derived from collection of separate domain specific types/objects.

## Typical Scenario

DataGrid view can be resolved from several partial types at runtime. In this scenario, developer can maintain the separation of partial classes in domain specific implementations and resolve single DataGrid canonical view object at runtime for frontend UI purposes.

## Early Code Examples

`Billing Customer` and `CRM Customer` will be configured in container extension as partial types, and resolve for a single DataGrid canonical object at runtime. Please note: final porting from current solution may change the code example given below)

The below code sample illustrates the use of the extension without using Canonical Object Lifetime manager.

```
IUnityContainer container = new UnityContainer();

container
    .AddNewExtension<CanonicalObject>()
    .Configure<CanonicalObject>
        .RegisterPartialType<IDataGridView<IBillingCust, BillingCust>>("CustomerView")
        .RegisterPartialType<IDataGridView<ICRMCustomer, CRMCustomer>>("CustomerView");

DataGridView customerView = container.Resolve<IDataGridView>("CustomerView");
```

The second code sample illustrates the use of extension with Canonical Object Lifetime Manager (COLifetime) that enables the singleton pattern for canonical object parts.

```
IUnityContainer container = new UnityContainer();

container
    .AddNewExtension<CanonicalObject>()
    .Configure<CanonicalObject>
        .RegisterPartialType<IDataGridView<IBillingCust, BillingCust>>("CustomerView", new COLifeTime())
        .RegisterPartialType<IDataGridView<ICRMCustomer, CRMCustomer>>("CustomerView", new COLifeTime());

DataGridView custView = Container.Resolve<IDataGridView>("CustomerView");
```

After use of the DataGridView customerView canonical object developer can resolve the partial object from canonical object/type by just resolving each of the partial object/type.

```
IBillingCust billingCust =
    container.Resolve<IBillingCust>("CustomerView");
```