Using the SAO Self Aware Object library

Project location: https://github.com/ScottLilly/DotNetSelfAwareObjects

Contact e-mail: github@scottlilly.com

Contents

What is it?

Self Aware Objects (SAO) is a way for you to define default values and validation rules for properties within your classes.

Why would I want to use it?

Do you repeatedly write code to do the following?

- Make sure your string properties aren't empty.
- Verify that your numeric properties are between certain values.
- Ensure that your string properties aren't too long, or too short.
- Assign default values to your properties.
- And so on

Then, when a new programmer joins your team, it takes them a while to find all your validation (especially if you're putting it in a controller, like you generally do with the Model-View-Controller design pattern).

If so, consider using the SAO library.

It will let you…

- Assign default values that are automatically populated when you instantiate an object.
- Declare the validation rules in your business class, right next to the properties.
- Perform complete validation of your business object, at any time, with a single call to its Validate() method (no controller class needed).

How can I use it?

1. Download the SAO project from here and include it in your solution.
2. Create your business class, using SAObject as its base class.
3. Apply the appropriate attributes to your properties.

That's it.

Here's how a sample class would look:

```
using System;

using SAO;
using SAO.Attributes.Property.Default;
using SAO.Attributes.Property.Validation;

namespace Engine.Demo
{
    public class Customer : SAObject
    {
        [DefaultEmptyGuidToNewGuid]
        public Guid ID { get; set; }

        [CannotBeEmptyString("Name cannot be blank")]
        public string Name { get; set; }

        [MinimumLength(5, "Address cannot be less than 5 characters")]
        [MaximumLength(100, "Address cannot be more than 100 characters")]
        public string Address { get; set; }

        [MinimumLength(2, "City cannot be less than 2 characters")]
        [MaximumLength(100, "City cannot be more than 100 characters")]
        public string City { get; set; }

        [ExactLengthString(2, "State must be two characters")]
        [ContainsOnlyLetters("State can only contain letters")]
        public string StateCode { get; set; }

        [MatchesRegEx(RegExConstants.CountrySpecific.US.ZIP_CODE, "ZIP Code
must be formatted like '99999' or '99999-9999'")]
        public string ZIPCode { get; set; }

        [MinimumValue(1, "Age must be larger than 0")]
        [MaximumValue(150, "Age cannot be larger than 150")]
        public int Age { get; set; }
    }
}
```

When you instantiate a Customer object, it will have a Guid assigned to the ID property, due to the [DefaultEmptyGuidToNewGuid] attribute.

When you get to the point where you want to validate the properties on this object, you just need to call the Validate() method. This will set the object's boolean IsValid value. If there are any errors, you can read the error messages from the object's ErrorMessage property (a list of strings).

Can I add my own default and validation attributes?

Yes.

To set up your own default attribute, you just need to implement the ISAOPropertyDefaultApplicator interface.

ISAOPropertyDefaultApplicator.cs

```
namespace SAO.Attributes.Interfaces
{
    public interface ISAOPropertyDefaultApplicator
    {
        bool NeedsDefaultApplied(object property);
        object DefaultValue { get; }
    }
}
```

Here's what one of the existing default applicators looks like:

DefaultEmptyGuidToNewGuidAttribute.cs

```
using System;

using SAO.Attributes.Interfaces;

namespace SAO.Attributes.Property.Default
{
    [AttributeUsage(AttributeTargets.Property, Inherited = true,
AllowMultiple = false)]
    public class DefaultEmptyGuidToNewGuidAttribute : Attribute,
ISAOPropertyDefaultApplicator
    {
        public object DefaultValue { get { return Guid.NewGuid(); } }

        public bool NeedsDefaultApplied(object property)
        {
            if(property is Guid)
            {
                return ((Guid)property == Guid.Empty);
            }

            return false;
        }
    }
}
```

Your own validation rules need to implement the ISAOPropertyValidator interface.

ISAOPropertyValidator.cs

```
namespace SAO.Attributes.Interfaces
{
    public interface ISAOPropertyValidator
    {
        void Validate(SAObject obj, object property);
    }
}
```

Here's the existing code for a validator that doesn't allow a property to be a null or empty string.

CannotBeEmptyStringAttribute.cs

```
using System;

using SAO.Attributes.Base;

namespace SAO.Attributes.Property.Validation
{
    [AttributeUsage(AttributeTargets.Property, Inherited = true,
AllowMultiple = true)]
    public class CannotBeEmptyStringAttribute :
SAOBasePropertyValidationAttribute
    {
        public CannotBeEmptyStringAttribute(string errorMessage) :
base(errorMessage)
        {
        }

        public override void Validate(SAObject obj, object property)
        {
            if((property == null) || (String.IsNullOrWhiteSpace(property as
string)))
            {
                obj.Invalidate(ErrorMessage);
            }
        }
    }
}
```

Can I use databinding with an SAObject?

Yes.

SAObject implements INotifyPropertyChanged and has an OnPropertyChanged event handler.

To bind the Name property, in the example Customer class above, just give it a backing variable (_name) and change it to the following:

```
private string _name;

[CannotBeEmptyString("Name cannot be blank")]
public string Name
{
    get { return _name; }
    set
    {
        if(_name != value)
        {
            _name = value;
            OnPropertyChanged("Name");
        }
    }
}
```

What are the existing defaults and validation rules?

Defaults

- DefaultEmptyGuidToNewGuid
- DefaultNullOrEmptyString
- DefaultNullValueTo

Validation rules

- CannotBeEmptyString
- CannotBeNull
- ContainsOnlyLetters
- ExactLengthString
- MatchesRegEx
- MaximumLength
- MaximumValue
- MinimumLength
- MinimumValue