Business Objects, Validation And Exceptions

Asked 16 years, 3 months ago Modified 15 years, 3 months ago

Viewed 20k times

38

I've been reading a few questions and answers regarding exceptions and their use. Seems to be a strong opinion that exceptions should be raised only for exception, unhandled cases. So that lead me to wondering how validation works with business objects.

Lets say I have a business object with getters/setters for the properties on the object. Let's say I need to validate that the value is between 10 and 20. This is a business rule so it belongs in my business object. So that seems to imply to me that the validation code goes in my setter. Now I have my UI databound to the properties of the data object. The user enters 5, so the rule needs to fail and the user is not allowed to move out of the textbox. . The UI is databound to the property so the setter is going to be called, rule checked and failed. If I raised an exception from my business object to say the rule failed, the UI would pick that up. But that seems to go against the preferred usage for exceptions. Given that it's a setter, you aren't really going to have a 'result' for the setter. If I set another flag on the object then that would imply the UI has to check that flag after each UI interaction.

So how should the validation work?

Edit: I've probably used an over-simplified example here. Something like the range check above could be handled easily by the UI but what if the valdation was more complicated, e.g. the business object calculates a number based on the input and if that calculated number is out of range it should be recjected. This is more complicated logic that should not be in th UI.

There is also the consideration of further data entered based on a field already entered. e.g.I have to enter an item on the order to get certain informaion like stock on hand, current cost, etc. The user may require this information to make decisions on further entry (liek how many units to order) or it may be required in order for further validation to be done. Should a user be able to enter other fields if the item isn't valid? What would be the point?

c# validation exception business-objects

Share

Improve this question

Follow

edited Sep 18, 2008 at 4:14

asked Sep 17, 2008 at 23:04

user11355

531 ● 1 ● 8 ● 8

2 All kinds of wonderful theory, little practical advice. Mike Thompson is on the right track, but is making a big assumption. Code would be helpful, or at least some idea of whether we're talking about ASP.Net or what. Just tagging as C# doesn't narrow it down much. – Dustman Sep 18, 2008 at 5:56

18 Answers

Sorted by: Highest score (default)

18

You want to delve a bit in the remarkable work of Paul Stovell concerning data validation. He summed up his ideas at one time in this article. I happen to share his point of view on the matter, which I implemented in my own libraries.

Here are, in Paul's words, the cons to throwing exceptions in the setters (based on a sample where a `Name` property should not be empty) :

> - There may be times where you actually need to have an empty name. For example, as the default value for a *"Create an account"* form.
> - If you're relying on this to validate any data before saving, you'll miss the cases where the data is already invalid. By that, I mean, if you load an account from the database with an empty name and don't change it, you might not ever know it was invalid.

> - If you aren't using data binding, you have to write a lot of code with `try/catch` blocks to show these errors to the user. Trying to show errors on the form as the user is filling it out becomes very difficult.
> - I don't like throwing exceptions for non-exceptional things. A user setting the name of an account to *"Supercalafragilisticexpialadocious"* isn't an exception, it's an error. This is, of course, a personal thing.
> - It makes it very difficult to get a list of all the rules that have been broken. For example, on some websites, you'll see validation messages such as *"Name must be entered. Address must be entered. Email must be entered"*. To display that, you're going to need a lot of `try/catch` blocks.

And here are basic rules for an alternative solution :

> 1. There is nothing wrong with having an invalid business object, so long as you don't try to persist it.
> 2. Any and all broken rules should be retrievable from the business object, so that data binding, as well as your own code, can

> see if there are errors and handle them appropriately.

Share Improve this answer Follow

answered Sep 25, 2008 at 12:35

Mac
8,309 ● 5 ● 42 ● 51

3 For a specific code solution to complex (exception-less) validation approach that supports `IDataErrorInfo` for auto-validation, try this: stackoverflow.com/questions/1721327/… – Matt Kocaj Nov 19, 2009 at 12:22

9

Assuming that you have separate validation and persist (i.e. save to database) code, I would do the following:

1. The UI should perform validation. Don't throw exceptions here. You can alert the user to errors and prevent the record from being saved.
2. Your database save code should throw invalid argument exceptions for bad data. It makes sense to do it here, since you cannot proceed with the database write at this point. Ideally this should never happen since the UI should prevent the user from saving, but you still need it to ensure database consistency. Also you might be calling this code from something other than the UI (e.g. batch updates) where there is no UI data validation.

Share Improve this answer

answered Sep 18, 2008 at 2:14

Follow

Mike Thompson
6,738 ● 3 ● 33 ● 39

3 Good point. The UI should not consider it an exception, since it was just a harmless mistake by the user, but the data access layer *should*, since it should never have gotten that far. – jeremcc Sep 18, 2008 at 5:11

8

I've always been a fan of Rocky Lhotka's approach in the CSLA framework (as mentioned by Charles). In general, whether it's driven by the setter or by calling an explicit Validate method, a collection of BrokenRule objects is maintained internally by the business object. The UI simply needs to check an IsValid method on the object, which in turn checks the number of BrokenRules, and handle it appropriately. Alternatively, you could easily have the Validate method raise an event which the UI could handle (probably the cleaner approach). You can also use the list of BrokenRules to display error messages to the use either in summary form or next to the appropriate field. Although the CSLA framework is written in .NET, the overall approach can be used in any language.

I don't think throwing an Exception is the best idea in this case. I definitely follow the school of thought that says Exceptions should be for exceptional circumstances, which a simple validation error is not. Raising an OnValidationFailed event would be the cleaner choice, in my opinion.

By the way, I have never liked the idea of not letting the user leave a field when it is in an invalid state. There are so many situations where you might need to leave the field temporarily (perhaps to set some other field first) before going back and fixing the invalid field. I think it's just an unnecessary inconvenience.

Share Improve this answer Follow

edited Sep 18, 2008 at 4:39

answered Sep 18, 2008 at 4:33

jeremcc

8,733 ● 12 ● 46 ● 55

5

You might want to move the validation outside of the getters and setters. You could have a function or property called IsValid that would run all the validation rules. t would populate a dictionary or hashtable with all of the "Broken Rules". This dictionary would be exposed to the outside world, and you can use it to populate your error messages.

This is the approach that is taken in CSLA.Net.

Share Improve this answer Follow

answered Sep 18, 2008 at 3:38

Charles Graham

24.8k ● 14 ● 47 ● 56

Exceptions should not be thrown as a **normal** part of validation. Validation invoked from within business objects

4

is a last line of defense, and should only happen if the UI fails to check something. As such they can be treated like any other runtime exception.

Note that here's a difference between defining validation rules and applying them. You might want to define (ie code or annotate) your business rules in your business logic layer but invoke them from the UI so that they can handled in a manner appropriate to that particular UI. The manner of handling will vary for different UI's, eg form based web-apps vs ajax web-apps. Exception-on-set validation offers very limited options for handling.

Many applications duplicate their validation rules, such as in javascript, domain object constraints and database constraints. Ideally this information will only be defined once, but implementing this can be challenge and requires lateral thinking.

Share Improve this answer

Follow

answered Sep 18, 2008 at 0:29

Brad at Kademi

1,320 ● 8 ● 7

3

Perhaps you should look at having both client-side and server-side validation. If anything slips past the client-side validation you can then feel free to throw an exception if your business object would be made invalid.

One approach I've used was to apply custom attributes to business object properties, which described the validation rules. e.g.:

```
[MinValue(10), MaxValue(20)]
public int Value { get; set; }
```

The attributes can then be processed and used to automatically create both client-side and server-side validation methods, to avoid the problem of duplicating business logic.

Share Improve this answer

Follow

answered Sep 17, 2008 at 23:41

Matt Howells

41.3k ● 20 ● 85 ● 104

3

I'd definitely advocate both client and server-side validation (or validating at the various layers). This is especially important when communicating across physical tiers or processes, as the cost of throw exceptions becomes increasingly expensive. Also, the further down the chain you wait for validation, the more time is wasted.

As to use Exceptions or not for data validation. I think it's ok to use exception in process (though still not preferrable), but outside of process, call a method to validate the business object (eg before saving) and have the method return the success of the operation along with any validation **errors**. Errors arent' exceptional.

Microsoft throw exceptions from business objects when validation fails. At least, that's how the Enterprise Library's Validation Application Block works.

```
using Microsoft.Practices.EnterpriseLibrary.Validation
using Microsoft.Practices.EnterpriseLibrary.Validation
public class Customer
{
  [StringLengthValidator(0, 20)]
  public string CustomerName;

  public Customer(string customerName)
  {
    this.CustomerName = customerName;
  }
}
```

Share Improve this answer

Follow

answered Sep 17, 2008 at 23:50

Rob Gray

3,266 ● 4 ● 36 ● 34

3

Your business objects should throw exceptions for bad inputs, but those exceptions should never be thrown in the course of a normal program run. I know that sounds contradictory, so I shall explain.

Each public method should validate its inputs, and throw "ArgumentException"s when they are incorrect. (And private methods should validate their inputs with "Debug.Assert()"s to ease development, but that's another story.) This rule about validating inputs to public methods (and properties, of course) is true for every layer of the application.

The requirements of the software interface should be spelled out in the interface documentation, of course, and it is the job of the calling code to make sure the

arguments are correct and the exceptions will never be thrown, which means the UI needs to validate the inputs before handing them to the business object.

While the rules given above should almost never be broken, sometimes business object validation can be very complex, and that complexity shouldn't be foisted onto the UI. In that case it's good for the BO's interface to allow some leeway in what it accepts and then provide for an explicit Validate(out string[]) predicate to check the properties and give feedback on what needs to be changed. But notice in this case that there are still well-defined interface requirements and no exceptions need ever be thrown (assuming the calling code follows the rules).

Following this latter system, I almost never do early validation on property setters, since that soft-of complicates the use of the properties, (but in the case given in the question, I might). (As an aside, please don't prevent me from tabbing out of a field just because it has bad data in it. I get clausterphobic when I can't tab around a form! I'll go back and fix it in a minute, I promise! OK, I feel better now, sorry.)

Share Improve this answer

Follow

answered Sep 18, 2008 at 3:40

Jeffrey L Whitledge

59.4k ● 9 ● 74 ● 100

It depends on what sort of validation you will be performing and where. I think that each layer of the

3

application can be easily protected from bad data and its too easy to do for it not to be worth it.

Consider a multi-tiered application and the validation requirements/facilities of each layer. The middle layer, Object, is the one that seems to be up for debate here.

- **Database**
  protects itself from an invalid state with column constraints and referential integrity, which will cause the application's database code to throw exceptions
- **Object**
  ?
- **ASP.NET/Windows Forms**
  protects the form's state (not the object) using validator routines and/or controls *without* using exceptions (winforms does not ship with validators, but there's an excellent series at msdn describing how to implement them)

Say you have a table with a list of hotel rooms, and each row has a column for the number of beds called 'beds'. The most sensible data type for that column is an unsigned small integer*. You also have a plain ole object with an Int16* property called 'Beds'. The issue is that you can stick -4555 into an Int16, but when you go to persist the data to a database you're going to get an Exception. Which is fine - my database shouldn't be allowed to say that a hotel room has less than zero beds, because a hotel room *can't* have less than zero beds.

* If your database can represent it, but let's assume it can
* I know you can just use a ushort in C#, but for the purpose of this example, let's assume you can't

There's some confusion as to whether objects should represent your business entity, or whether they should represent the state of your form. Certainly in ASP.NET and Windows Forms, the form is perfectly capable of handling and validating its own state. If you've got a text box on an ASP.NET form that is going to be used to populate that same Int16 field, you've probably put a RangeValidator control on your page which tests the input before it gets assigned to your object. It prevents you from entering a value less than zero, and probably prevents you from entering a value greater than, say, 30, which hopefully would be enough to cater for the worst flea-infested hostel you can imagine. On postback, you would probably be checking the IsValid property of the page *before* building your object, thereby preventing your object from ever representing less than zero beds and preventing your setter from ever being called with a value it shouldn't hold.

But your object is still *capable* of representing less than zero beds, and again, if you were using the object in a scenario not involving the layers which have validation integrated into them (your form and your DB) you're outta luck.

Why would you ever be in this scenario? It must be a pretty *exceptional* set of circumstances! Your setter

therefore needs to throw an exception when it receives invalid data. It should never be thrown, but it could be. You could be writing a Windows Form to manage the object to replace the ASP.NET form and forget to validate the range before populating the object. You could be using the object in a scheduled task where there is no user interaction at all, and which saves to a different, but related, area of the database rather than the table which the object maps to. In the latter scenario, your object can enter a state where it is invalid, but you won't know until the results of other operations start to be affected by the invalid value. If you're checking for them and throwing exceptions, that is.

Share Improve this answer

Follow

answered Sep 18, 2008 at 4:59

Shabbyrobe

12.6k ● 15 ● 62 ● 88

3

I tend to believe business objects should throw exceptions when passed values that violate its business rules. It however seems that winforms 2.0 data binding architecture assumes the opposite and so most people are rail-roaded into supporting this architecture.

I agree with shabbyrobe's last answer that business objects should be built to be usable and to work correctly in multiple environments and not just the winforms environment, e.g., the business object could be used in a SOA type web service, a command line interface,

asp.net, etc. The object should behave correctly and protect itself from invalid data in all these cases.

An aspect that is often overlooked is also what happens in managing the collaborations between objects in 1-1, 1-n or n-n relationships, should these also accept the addition of invalid collaborators and just maintain a invalid state flag which should be checked or should it actively refuse to add invalid collaborations. I have to admit that I'm heavily influenced by the Streamlined Object Modeling (SOM) approach of Jill Nicola et al. But what else is logical.

The next thing is how to work with windows forms. I'm looking at creating a UI wrapper for the business objects for these scenarios.

Share Improve this answer

Follow

answered Sep 25, 2008 at 12:08

Jide Ogundipe

3

As Paul Stovell's article mentioned, you can implement error-free validation in your business objects by implementing the IDataErrorInfo interface. Doing so will allow user error notification by WinForm's ErrorProvider and WPF's binding with validation rules. The logic to validate your objects properties is stored in one method, instead of in each of your property getters, and you do not necessarily have to resort to frameworks like CSLA or Validation Application Block.

As far as stopping the user from changing focus out of the textbox is concerned: First of all, this is usually not the best practice. A user may want to fill out the form out of order, or, if a validation rule is dependent on the results of multiple controls, the user may have to fill in a dummy value just to get out of one control to set another control. That said, this can be implemented by setting the Form's `AllowValidate` property to its default, `EnableAllowFocusChange` and subscribing to the Control.Validating event:

```
private void textBox1_Validating(object sender, Ca
{
    if (textBox1.Text != String.Empty)
    {
        errorProvider1.SetError(sender as Control,
        e.Cancel = true;
    }
    else
    {
        errorProvider1.SetError(sender as Control,
    }
}
```

Using rules stored in the business object for this validation is a little more tricky since the Validating event is called before the focus changes and the data bound business object is updated.

Share Improve this answer

Follow

answered Dec 29, 2008 at 17:09

foson

10.2k ● 2 ● 37 ● 53

1

You might like to consider the approach taken by the Spring framework. If you're using Java (or .NET), you can use Spring as-is, but even if you're not, you could still use that pattern; you'd just have to write your own implementation of it.

Share Improve this answer

Follow

answered Sep 17, 2008 at 23:17

Andrew Swan

13.6k ● 23 ● 73 ● 99

1

Throwing an exception in your case is fine. You can consider the case a true exception because something is trying to set an integer to a string (for example). The business rules lack of knowledege of your views means that they should consider this case exceptonal and return that back to the view.

Whether or not you validate your input values before you send them through to the business layer is up to you, I think that as long as you follow the same standard throughout your application then you will end up with clean and readable code.

You could use the spring framework as specified above, just be careful as much of the linked document was indicating writing code that is not strongly typed, I.E. you may get errors at run time that you could not pick up at compile time. This is something I try to avoid as much as possible.

The way we do it here currently is that we take all the input values from the screen, bind them to a data model object and throw an exception if a value is in error.

Share Improve this answer

Follow

answered Sep 17, 2008 at 23:38

Odd

4,757 ● 6 ● 32 ● 27

1

In my experience, validation rules are seldom universal across all screens/forms/processes in an application. Scenarios like this are common: on the add page, it may be ok for a Person object not to have a last name, but on the edit page it must have a last name. That being the case I've come to believe that validation should happen outside of an object, or the rules should be injected into the object so the rules can change given a context. Valid/Invalid should be an explicit state of the object after validation or one that can be derived by checking a collection for failed rules. A failed business rule is not an exception IMHO.

Share Improve this answer

Follow

answered Sep 18, 2008 at 0:24

Daniel Auger

12.6k ● 5 ● 53 ● 73

1

Have you considered raising an event in the setter if the data is invalid? That would avoid the problem of throwing an exception and would eliminate the need to explicitly check the object for an "invalid" flag. You could even pass

an argument indicating which field failed validation, to make it more reusable.

The handler for the event should be able to take care of putting focus back onto the appropriate control if needed, and it could contain any code needed to notify the user of the error. Also, you could simply decline to hook up the event handler and be free to ignore the validation failure if needed.

Share Improve this answer

Follow

answered Sep 18, 2008 at 3:21

Jeromy Irvine

11.8k ● 3 ● 41 ● 53

0

I my opinion this is an example where throwing an exception is okay. Your property probably does not have any context by which to correct the problem, as such an exception is in order and the calling code should handle the situation, if possible.

Share Improve this answer

Follow

answered Sep 17, 2008 at 23:12

Kimoz

375 ● 4 ● 11

0

If the input goes beyond the business rule implemented by the business object, I'd say it's a case not handled by the busines object. Therefore I'd throw an exception. Even though the setter would "handle" a 5 in your example, the business object won't.

For more complex combinations of input, a vaildation method is required though, or else you'll end up with quite complex validations scattered about all over the place.

In my opinion you'll have to decide which way to go depending on the complexity of the allowed/disallowed input.

Share Improve this answer

Follow

answered Sep 17, 2008 at 23:15

Ludvig A. Norin

5,325 ● 7 ● 33 ● 34

0

I think it depends on how much your business model is important. If you want to go the DDD way, your model is the most important thing. Therefore, you want it to be in a valid state at all time.

In my opinion, most people are trying to do too much (communicate with the views, persist to the database, etc.) with the domain objects but sometimes you need more layers and a better separation of concerns i.e., one or more View Models. Then you can apply validation without exceptions on your View Model (the validation could be different for different contexts e.g., web services/web site/etc.) and keep exception validations inside your business model (to keep the model from being corrupted). You would need one (or more) Application Service layer to map your View Model with your Business Model. The business objects should not be polluted with validation attributes often related to specific frameworks e.g., NHibernate Validator.

Share Improve this answer

Follow

answered Aug 27, 2009 at 0:51

W3Max

3,406 ● 5 ● 37 ● 61