When and why you should use void (instead of e.g. bool/int)

30

10

I occasionally run into methods where a developer chose to return something which isn't critical to the function. I mean, when looking at the code, it apparently works just as nice as a `void` and after a moment of thought, I ask "Why?" Does this sound familiar?

Sometimes I would agree that most often it is better to return something like a `bool` or `int`, rather then just do a `void`. I'm not sure though, in the big picture, about the pros and cons.

Depending on situation, returning an `int` can make the caller aware of the amount of rows or objects affected by the method (e.g., 5 records saved to MSSQL). If a method like "InsertSomething" returns a boolean, I can have the method designed to return `true` if success, else `false`. The caller can choose to act or not on that information.

something, it tells you that the method is of a type you **have** to do something with the returned result.

- Another issue would be, if the method implementation is unknown to you, what did the developer decide to return that isn't function critical? Of course you can comment it.
- The return value has to be processed, when the processing could be ended at the closing bracket of method.
- What happens under the hood? Did the called method get `false` because of a thrown error? Or did it return false due to the evaluated result?

What are your experiences with this? How would you act on this?

c# programming-practices language-discussion

edited Jun 29 '16 at 3:13
christophos
20 5

asked Apr 13 '11 at 9:15
Independent
475 1 4 21

1 A function that returns void isn't technically a function at all. It's just a method/procedure. Returning `void` at least lets the developer know that the method's return value is inconsequential; it does an action, rather than computing a value. – KChaloux Nov 15 '12 at 15:09

5 Answers

32

In the case of a bool return value to indicate the success or failure of a method, I prefer the `Try` -prefix paradigm used in various in .NET methods.

For example a `void InsertRow()` method could throw an exception if there already exists a row with the same key. The question is, is it reasonable to assume that the caller ensures their row is unique before calling InsertRow? If the answer is no, then I'd also provide a `bool TryInsertRow()` which returns false if the row already exists. In other cases, such as db connectivity errors, TryInsertRow could still throw an exception, assuming that maintaining db connectivity is the caller's responsibility.

edited Jul 10 '15 at 3:38
Suamere
600 1 7 22

answered Apr 13 '11 at 9:36
Bubblewrap
974 8 9

Absolutely a +1 for trying to invent a principle for TryXX methods which apparently make both try method and main method easier to maintain and easier to understand their purpose. – Independent Apr 13 '11 at 10:41

+1 Well said. The key to the best answer for this question is that sometimes you want to give the caller the option of an assertive method which will throw an exception. In these cases, though, the exception should not be caught and fake-handled by the assertive method. The point is that the caller is made responsible. On the other hand, a Try-paradigm (or Monad) should catch and handle exceptions, then either pass back a bool, or a descriptive Enum if more detail is necessary. Note that a caller expects that a Try/Monad will NEVER throw an exception. It's like an entry point. – Suamere Jul 10 '15 at 0:39

So since an exception is expected to never occur from a Try/Monad, yet a bool is not descriptive enough to inform the caller that a db connection failed, or an http request has one of many return values that are necessary to act on differently, you can use an Enum instead of a bool for your Try/Monad. – Suamere Jul 10 '15 at 0:44

If you have the TryInsertRow - returning a bool - if, say, there is more than one unique constraint in the database - how do you know exactly what the error was - and communicate that to the user? Should a richer return type be used containing the error message / code & success bool? – niico Aug 13 '16 at 11:00

28

IMHO returning a "status code" stems from historical times, before exceptions became commonplace in mid- to higher level languages such as C#. Nowadays it is better to throw an exception if some unexpected error prevented your method from succeeding. That way it is sure the error doesn't go unnoticed, and the caller can deal with it as appropriate. So in these cases, it is perfectly fine for a method to return nothing (i.e. `void`) instead of a boolean status flag or an integer status code. (Of course one should document what exceptions the method may throw and when.)

On the other hand, if it is a function in the strict sense, i.e. it performs some calculation based on input parameters, and it is expected to return the result of that calculation, the return type is obvious.

There is a gray area between the two, when one *may* decide to return some "extra" information from the method, if it is deemed useful for its callers. Like your example about the number of affected rows in an insert. Or for a method to put an element into an associative collection, it may be useful to return the previous value associated with that key, if there was any. Such uses can be identified by carefully analyzing the (known and anticipated) usage scenarios of an API.

edited Sep 24 '14 at 20:25
Dan
317 1 8

answered Apr 13 '11 at 9:25
Péter Török
44.3k 15 151 182

4 +1 for exceptions over status code. The exception (bad pun intended) to this is the well used TryXX pattern. – Andy Lowry Apr 13 '11 at 9:29

éter I'm with you there. TcF and not silence down errors! Probably, though, the grey zone. There **may** always be of use return something. Affected rows,

– ashes999 Apr 13 '11 at 17:39

2 -1 Wow, your own words are so contradictory and wrong. You should never use exceptions for control flow or communication. They are infinitely more expensive than a conditional. When is the time "before exceptions became commonplace"?... you mean before try/catch blocks have become commonplace? Exceptions have been common since forever. "Throw an exception if some unexpected error..." How do you take action on something that is unexpected? Why would you catch an exception, then re-throw it? That is soo expensive. Why do you say "error"? Errors and Exceptions are different things. – Suamere Jul 10 '15 at 0:17

1 And who's to say a thrown exception won't go unnoticed? Nothing forces anybody to log in a catch block, why not log in a "then" block? Why "document what exceptions the method may throw and when"? How about if we write self-documenting code, and a method simply returns an enum with the expected end-states of that method? If a possible exception is avoidable by checking state, they should be caught and handled without throwing. – Suamere Jul 10 '15 at 0:17

14

Peter's answer covers the exceptions well. Other consideration here involve Command-Query Separation

The CQS principle says that a method should either be a command, or a query and not both. Commands should never return a value, only modify the state, and queries should only return and not modify the state. This keeps the semantics very clear and help make code more readable and maintainable.

There are a few cases were violating the CQS principle is a good idea. These are usually around performance or thread safety.

edited Apr 13 '11 at 9:41

Péter Török
44.3k 15 151 182

answered Apr 13 '11 at 9:34

Andy Lowry
1,772 2 16 16

1 -1 Wrong and Wrong. Firstly, the whole point of CQS lies in the Q. A caller should be able to get calculations or external calls through some stateful service without fear of altering that service's state. Also, while CQS is a helpful principle to loosely follow, it is only strictly followed in specific designs. Lastly, even in strict CQS, nothing says a command can't return a value, it says it should not be calculating or calling external calculations and returning *data*. Either C or Q can feel free to return their end-state, if that is useful for the caller. – Suamere Jul 10 '15 at 0:31

Ah but number of rows affected by a Command makes building new commands from old ones much easier. Source: years and years of experience. – Joshua Jan 1 '17 at 23:04

@Joshua: Likewise, "add new record and return an ID (for some structures, a row number) that was generated during the add" can be used for purposes which cannot otherwise be achieved efficiently. – supercat Mar 8 '17 at 15:19

You shouldn't need to return the rows affected. The command should know how many will be affected. If it is anything but that #, it should roll back and throw an exception since something weird just happened. Thus, that is information the caller doesn't really care about it (unless user needs to know the

3

Whatever the path is your going to walk with this. Don't have return values in there for future use (e.g. have functions always return true) this is a terrible waste of effort and doesn't make the code clearer to read. When you have a return value you should always use it, and to be able to use it you should have at least have 2 possible return values.

answered Apr 13 '11 at 10:02

refro
1,171 5 15

+1 This doesn't answer the question, **but** it is definitely correct information. When making a decision, the decision should be based on an a need. If the callers don't find the return data useful, it shouldn't be done. And nobody finds use in returning true 100% of the time for "future proofing". Granted, if "future" is like... a couple days from now and there's a task existing for it, that's a different story, lol. – Suamere Jul 10 '15 at 0:33

1

I used to write lots of void functions. But since I got on the whole method chaining crack, I started returning this rather than void -- might as well let someone take advantage of the return cause you can't do squat with void. And if they don't want to do anything with it, they can just ignore it.

answered Apr 13 '11 at 13:21

Wyatt Barnett
20k 45 68

1 Method chaining can make inheritance cumbersome and difficult though. I wouldn't do method chaining unless you have a good reason to do it other than not wanting to give your methods "void". – KallDrexx Apr 13 '11 at 16:57

True. But inheritance can be cumbersome and difficult to use. – Wyatt Barnett Apr 13 '11 at 19:10

Looking at a unknown code which return whatever (mentioned the whole object) pays much attention just check the flow inside and outside methods.. – Independent Apr 16 '11 at 9:28

I don't suppose you got into `Ruby` at some point? That's what introduced me to method chaining. – KChaloux Nov 15 '12 at 15:12

One qualm I have with method chaining is that it makes it less clear whether a statement like `return Thing.Change1().Change2();` is expecting to change `Thing` and return a reference to the original, or is expected to return a reference to a new thing which differs from the original, while leaving the original untouched. – supercat Aug 23 '14 at 22:56