Here are my follow-up answers to the questions. If I misunderstood your question, didn’t answer it completely, or if you can think of a different answer, please let me know by leaving a comment at <http://scottlilly.com/FIWebinar>.

* **How would you have default values and avoid too lengthy code ? Let's say for example that 90% of the time you would include all categories so I would not want to repeat that each time I use the FI**

You could make an instantiating function named „CreateStandardSalesReport()“ (for example). Inside that function, it would call the private constructor and set the default values. Its returning datatype would be an interface that is farther in the chain, past the functions for variables that were automatically set inside CreateStandardSalesReport.

Here is an example of how that might be done: <https://gist.github.com/ScottLilly/85091b9f61e66256a69a7909a05337fd>

I would change the interfaces, so the functions that can be skipped over are first in the „chain“. It’s easier to skip over the first five functions (for example), than to create interfaces and functions that let you optionally skip over the first two functions, then the fifth function, then the seventh function.

You might also want to integrate the Builder pattern, as mentioned in one of the questions below.

* **how does this compare to the "Curiously Recurring Template Pattern"? such as expressed here** <http://zpbappi.com/curiously-recurring-template-pattern-in-csharp/>

That pattern is an interesting way to do method chaining. Although, it looks like you will still need to create individual interfaces, to enforce any grammar rules.

* **Can't you just use annotations to require a certain order and ensure parameters contain data?**

Yes, you could use annotations on an entity, to ensure the required properties were set, before being able to execute a function. However, another developer could forget to set a property value, and the error will only be detected at runtime.

With the fluent interface pattern, other developers will have the additional help of IntelliSense, to lead them through the chain of required functions to call. They could still pass an invalid parameter to a function. However, they would not be able to skip over calling the function.

* **Can we use Builder pattern where we create different set of Builder class for different values of Report properties?**

Yes, you could combine fluent interfaces with the Builder pattern. That would be a good way to handle a situation where you have several common ways to set the values for the some of the chaining functions.

For example, if Accounting reports should always call IncludeAllSalesPeople, IncludeAllCategories, ExcludeReturnedOrders, and IncludeUnshippedOrders, that could have one Builder class that calls those functions. You could have a different Builder class to set the values to only include all the categories for items that are physical products (and not downloadable items), for the Shipping department.

* **I would like to know about many Id's inside the filter, how to deal with this?**

If I was building the ReportGenerator class for a real program, I would probably have a function for passing in a list of Salesperson IDs (as if the function was receiving a list of checked items in a datagrid that displayed the salespeople).

Inside that function, I would add the IDs from that passed parameters into the private \_includedSalespoersonIDs variable, if that ID was not already in the list.

* **Is this the only way to implement fluent interfaces? If not, what are the other approaches and how are they different from your approach?**

This is the only way I’ve used. You might be able do something similar with extension methods that only work for specific datatypes (which would be the interfaces we use for the grammar). But, method seems less clear, to me.

If anyone is aware of a different method, please share it.

* **How would you implement the IncludeSalespersonID function within the report class or option that are mutually exclusive when the report is actually being built**

When you create your fluent interface’s grammar rules, you should design it to prevent mutually-exclusive functions. For example, in the ReportGenerator fluent interface, you can call “IncludeAllSalespeople”, or you can call “IncludeSalespersonID”. You can call “IncludeUnshippedOrders”, or you can call “ExcludeUnshippedOrders”. You can only call one, or the other – not both.

* **How exception handling works in fluent interface ? second , how it works with optional parameters ?**

Exceptions would be caught at runtime. You could add parameter validation, that could throw an exception if the passed parameter was invalid. Also, when the ending function is called (BuildReport or SendEmail, for example), it could throw an exception.

The fluent interface can ensure that, when other programmers user your class, they will call the all the required functions to set the required parameters. However, if you do not include other data validation, they could set the parameters to invalid values – for example, setting the “to” date before the “from” date, when specifying a data range.

* **Would it be possible to convert BDD scenarios to fluent?**

This seems like it could be a great idea. I haven’t used SpecFlow, but a fluent interface would almost match with creating FitNesse fixtures.

If you show the users (or business analysts) the concept of method chaining, they should be able to create the grammar for you, using business terms. Then, you could use that to build the required fluent interfaces.

This is definitely an idea I want to think about some more. It may be a great way to deal with correctly understanding complex business requirements.

* **If include All categories is optional, will the group by sort by will be available to continue with the BuildReport? so, the order would include a Build report method for all functions that are deemed optional?**

If you build the grammar to allow that, it should let you create that as a valid “chain”. I think the answer to the first question (see source code here: <https://gist.github.com/ScottLilly/85091b9f61e66256a69a7909a05337fd>) will show how to do that.

Please let me know if that sample doesn’t answer your question.

* **How to handle exception thrown by preceding method? How to stop chaining, or ignore error if not critical and continue chaining?**

The chaining functions only set the values of the private variables, and “return this”. So, there should not ever be an exception – unless you add your own data validation in those functions. In that case, when the code is run, and an invalid parameter is passed, it will throw whatever type of exception you specify, and stop executing.

You could put logic into the chaining functions that would check if the passed parameter is invalid. If it is, instead of having the function throw an exception, you could have it determine a good (or default) value to use.

* **Over time, lets say the ReportGenerator might add new funcionality. Is it possible to have more than one path to reach the ending function? And how can we ensure maybe though unit testing that the chain leads to an ending function?**

Yes, it is very possible to have more than one path reach the ending function. The ReportGenerator class has several possible paths (Include, or Exclude the returned and unshipped orders, for example).

Ensuring complete unit testing of all possible chains is interesting. If I was doing that, I’d probably to TDD, and use ReSharper (or some other static analysis tool) to show any functions in any interfaces that are not called. By looking at uncalled functions in the interfaces, that ***should*** inform us of any missing paths.

When I work on the fluent interface creating tool, that sounds like a good feature to add – automated generation of unit tests for each chain.

* **I think you just need to get the result of IncludeSalespersonId inside the foreach and continue from there to avoid the casting.**

I think this example is a little tricky, because you must either enter at least on salesperson ID (through IncludeSalespoersonID), or call IncludeAllSalespeople, before you can call one of the category-setting functions.

If you have an example that works, please share it at <http://scottlilly.com/FIWebinar>.

* **How would you go about baseclassing and extending this pattern with generics and inheritance is the essence of what i am trying to understand**

I have not tried to create a generic version of a fluent interface engine. Every time I’ve built a fluent interface, it has been very specific to one class (such as the ReportGenerator class).

If you needed to create “chains” for a class that were extremely different, you might want to have ReportGenerator as a base class, and create child classes, with their own interfaces that implement the different chains. For example, if you wanted to have a ManagementReportGenerator fluent interface for management reports, which might show different information, and have very different “chaining” options. You might also have an AccountingReportGenerator for the accountants, which might have a massively different fluent interface. Those would have their own sets of interfaces, but might use some functions from the BaseReportGenerator class.

* **I want to call one interface or a different one according to a value previously set**

**For example can this be possible? if IncludeSalesPersonID is called then AllCategories is selectable, and if they select allSalesPersons, then AllCategories is not enabled (to prevent a call from being too big)**

Yes, you could do this. When building the grammar spreadsheet, in the row for IcludeAllSalespeople, don’t put a „Y“ in the IncludeAllCategories. Then, your interface for that row might be named IcanSetOneCategoryID. That interface would be defined to only have one function „IncludeCategoryID“, and its return datatype would be „ICanAddCategoryIDOrSetReturnedOrdersInclusion“.