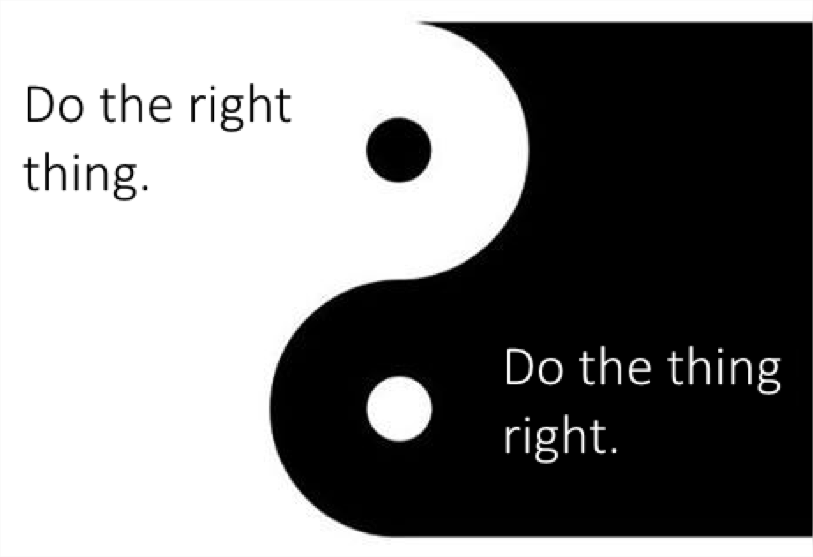
# [Development](https://confluence.schroders.com/display/GCFN/Development)

In the mission of improving code quality and reduce the time-pressure on testing (by enabling automated testing) it becomes necessary to establish a set of coding patterns and practices as far as Schroders development team is considered.

Aim for the sky but set your sights on the next step. The sky in this context are the [Clean Code Developer Principles](https://ccd_school.gitbooks.io/clean-code-developer-com/content/) (in white grade), the next step however just a few rule of high importance:

* [DRY: Don't Repeat Yourself](https://confluence.schroders.com/display/GCFN/Development#Development-DRY:Don'tRepeatYourself)
* [KISS: Keep It Simple, Stupid](https://confluence.schroders.com/display/GCFN/Development#Development-KISS:KeepItSimple,Stupid)
* [IOSP: Integration Operation Segregation Principle](https://confluence.schroders.com/display/GCFN/Development#Development-IOSP:IntegrationOperationSegregationPrinciple)
* [POLA: Principle of Least Astonishment](https://confluence.schroders.com/display/GCFN/Development#Development-POLA:PrincipleofLeastAstonishment)
* [SRP: Single Responsibility Principle](https://confluence.schroders.com/display/GCFN/Development#Development-SRP:SingleResponsibilityPrinciple)
* [SoC: Separation of Concerns](https://confluence.schroders.com/display/GCFN/Development#Development-SoC:SeparationofConcerns)
* [DIP: Dependency Inversion Principle](https://confluence.schroders.com/display/GCFN/Development#Development-DIP:DependencyInversionPrinciple)
* [Pathfinder Rule](https://confluence.schroders.com/display/GCFN/Development#Development-PathfinderRule)



The geeks of us may want to print out the [Clean Code Developer Cheat Sheet](https://www.planetgeek.ch/wp-content/uploads/2014/11/Clean-Code-V2.4.pdf)

DRY: Don't Repeat Yourself

**Why?**

Every duplication of both code and manual tasks fosters inconsistencies and mistakes.

DRY "*Don't Repeat Yourself*" is valid since the very begin of software development otherwise there wouldn't be subroutines or data normalization. Even though it is the most frequently disregarded principle. It is too simple to repeat code using copy & paste. Especially when it is meant to go quick this happens far too often.

Clean code developer practice in the red grade to regard this principle. They become aware when they repeat code or other artifacts. They recognize repetitions created by themselves or others. They clean-up repetitions by refactoring as long as no other principles or limitations contradict.

KISS: Keep It Simple, Stupid

**Why?**

Doing more than the obvious keeps customers waiting and complicates solutions unnecessarily.

To quote Albert Einstein: "Everything should be done as simple as possible, but not simpler". Prerequisite for evolvability of code is that code is comprehensible. Always prefer a simple, clear and easy to understand solution. It should ring an alarm if you won't understand you own code after a short period of time. Even more important is that other developers can understand the code quickly. Pair Programming and regular reviews support this. These provide also control if the simplest solution was indeed used.

Simplicity is also loved in wording and phrasing in specifications; plentiful usages of numerous synonyms makes understanding difficult; twisted phrases are eating time just like in: "A twin is asked the following: who is the son of your father, but it is not the brother of you?"

Technical details drive temptation to head for a complex solution. If the well-known and obvious appears too boring, a complex solution might sneak in. Provided the simple solution works, it shall be favored. Same so for data structures. If IEnumerable will be sufficient, ICollection or even IList shall not be used. At the same time, lists should not be used when dictionaries are better fitting in.

IOSP: Integration Operation Segregation Principle

**Why?**

A broad symptom of code which is hard to change are deep hierarchies of functional dependencies. The decrease understandability and hamper both automatized tests and refactoring.

If a method contains both logic and calls to other methods, the total behavior will be no longer clear. The instructions are blurred over a possibly very deep hierarchy. Furthermore this kind of method has a tendency to grow unlimited.

IOSP calls for a clear separation:

* Either a method contains exclusively logic, meaning transformations, control structures or API invocations. Then it's called an *Operation*.
* Or a method does not contain any logic but exclusively calls other methods within its code basis. Then it's called *Integration*.

That explicit separation has many positive effects:

1. Methods tend to stay short. More than 10, 20 or 30 lines of pure logic or exclusively method calls "feel strange". As mixing is not allowed small methods will be extracted.
2. Short logic methods are easy to test as they don't have dependencies.
3. Short logic methods are relatively easy to understand. The methods name can drive the meaning.
4. Short integration methods are very well understandable and reveal what happens at first glance.
5. Correctness of integrations can be reviewed easily. Basically just the process order needs to be double-checked. Compiler and unit-tests of operations do the rest.
6. Integrations can be nicely expanded by inserting additional method calls. Understandability stays.

Any developer willing can apply IOSP off-hand. Also IOSP compliance is trivial to verify as integration and operation differ notably in form.

POLA: Principle of Least Astonishment

**Why?**

If a component behaves surprisingly different than expected, your application will become unnecessarily complex and error prone.

To a high degree software development is a creative process. In that process it is important to be in the flow. In that state code easily splutters. Any flow disturbance is an interruption that results in less code being produced in the available time and code quality dropping. After each interruption the developer needs gathering speed to get into the flow again. Surprises are disturbances. They result in interruptions and mistakes. Sample: If keyboard shortcuts in the development environment the common short Ctrl-C has completely different meaning, this will disturb the developer. Using the "wrong" shortcut will annoy him or her again and again. This prevents creative work.

Software implementation should avoid surprises. If a method named GetValue() does not only return a value but also changes the systems state, a developer ideally would disregard that method to avoid naughty surprises. In worst case he won't be aware of that suspicious behavior. (Query methods which change the state defy the *Command Query Separation* principle). Test-driven development fosters least astonishing interfaces as interfaces are designed and implemented from the clients view.

SRP: Single Responsibility Principle

**Why?**

Focus eases understanding. A class having exactly one task is easier to understand than a convenience store.

Single Responsibility Principle (SRP) is one of the [SOLID](https://ccd_school.gitbooks.io/clean-code-developer-com/content/02grades/07_Orange_Grade.html#solid) principles stating that a calls shall have only one responsibility.

Beyond the Single Responsibility Principle stands the idea that changing or enhancing functionality should affect only a few classes. The more classes need to be changed the higher is the risk that problems arise in areas which basically have nothing to do with the enhancement. A SRP violation leads to coupling and increased complexity. It becomes harder to understand the code.

SoC: Separation of Concerns

**Why?**

If a code unit does not have a clear purpose, it will be difficult to understand, difficult to use, difficult to enhance and difficult to correct.

You should not combine multiple concerns in one class. What is a concern? Concerns a "completely different" purposes. Concern are considered to be orthogonal to each other and especially orthogonal to the main functionality of a functional unit. Samples for typical concerns are: Tracing, logging, transaction handling, caching. If you follow SoC principle, you will extract these concerns into specialized functional units.

The Separation of Concerns principle is closely connected to the Single Responsibility principle. Concerns are a superset of responsibilities. A responsibility ideally consists of exactly one concern which is its core functionality. Nevertheless more often multiple concerns are mingled in one responsibility. This usually cannot be totally avoided. So the principle doesn't say that a responsibility must consist of one concern but just that the concerns need to be separated. For instance within a method multiple concerns shall be clearly visible, so that concerns don't spread willfully over the whole method but are grouped in a way that makes clear what belongs to a certain concern.

Domain Driven Design attempts to strictly separate business domain from infrastructure. So a business domain class must not contain infrastructure like database access but can only represent business logic. Persistence is a "Concern" which has nothing to do with business logic. Separation of Concerns leads to lose coupling and high cohesion. Each component focusses on a single task its concern - and thereby is easy to understand. All parts of that component would be oriented on that task so that the part a closely connected (high cohesion). Separation of Concerns further leads to well testable components because a focused purpose of a code unit needs less wide tests. Regarding the code unit under test less parameter combinations have to be checked. To live SoC consequently Aspect Oriented Programming (AOP) needs to enhance Object Orientation. That way aspects like transaction handling, tracing or caching can be fully extracted from a method.

DIP: Dependency Inversion Principle

**Why?**

Class isolation is a prerequisite for testing to the point. Isolation is when classes receive no more implementation dependencies neither at compile time nor at runtime. Concrete dependencies shall be decided on as late as possible. Ideally at runtime.

Also Dependency Inversion Principle is a [SOLID](https://ccd_school.gitbooks.io/clean-code-developer-com/content/02grades/08_Yellow_Grade.html#solid) principle. It says:

* High-level classes must not depend on low-level classes but both from interfaces.
* Interfaces must not depend on details but details on interfaces.

If a high-level class uses a low-level class directly, there will be a strong coupling between both of them. You will hit problems latest when trying to do an isolated test of the high-level class. Therefore the high-level class shall depend on an interface which again is implemented by the low-level class. That way the low-level class can be mocked in a unit test.

Basically there are three options to resolve an inverted, abstract dependency at runtime:

* Constructor parameter
* Inversion of Control Container (IoC Container), e. g. Castle Windsor
* Dependency Lookup

In yellow grade dependencies are injected using constructor parameters. That is a simple solutions which works nicely with a small set of classes. Later on in [green grade](https://ccd_school.gitbooks.io/clean-code-developer-com/content/02grades/08_Yellow_Grade.html#grade-4-green) an IoC Container and Dependency Lookup will be used.

*Source: ObjectMentor; author: Robert C. Martin; Article regarding Dependency Inversion Principle; published 1996 in The C++ Report, Engineering Notebook*

Pathfinder Rule

Why?

A broad symptom of code which is hard to change are deep hierarchies of functional dependencies. The decrease understandability and hamper both automatized tests and refactoring.

* Evolvability: \*\*\*
* Correctness:
* Production Efficiency:
* Continuous Improvement:
* Single Developer

Establishing the CCD values needs time. Also it is difficult to apply the principles to the whole code base as a CCD rarely starts alone on a green field project. Therefore it is more realistic and more motivating to aim for tiny but continuous progress.

Accordingly the pathfinder rule belongs to the clean code development fundamentals. It says:\ *Leave the campground cleaner than you found it.*

Concerning software development that means: Clean code developers leave code in a better state than they found it. So after work accomplished code shall apply more to CCD values than before.

What exactly to be done is specific to situation and code and of course to the grade currently worked on. A CCD in read grade would for an instance move code into version control, if it wasn't yet in there. And he would focus on eliminating any kind of redundancies which are violations of the DRY principle.

So a CCD will steadily try to heal weaknesses in the sense of CCD values wherever he finds it. In small steps. And naturally he will try hard to avoid weaknesses upfront. As said: Always at the level of his personal development.

This maxim is essential in the evolution of a CCD bearing [broken windows theory](https://en.wikipedia.org/wiki/Broken_windows_theory) in mind. According to that decay of quality in general starts with trifles which are ignored long enough.

Following the pathfinder rule no "broken windows" will be produced and existing ones are repaired one by one. The pathfinder rule systematically eliminates fissures and bumps based on the CCD values, so that no further deposits are formed. This practice works proactively against code erosion which is so fundamental that it is included in red grade.

# [Patterns and Practices](https://confluence.schroders.com/display/GCFN/Patterns+and+Practices)

In this section developers collect examples on how common development activities can be efficiently implemented according to the guidelines.

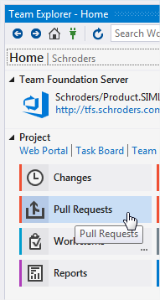
# [Code Review Guidelines and approved practices](https://confluence.schroders.com/display/GCFN/Code+Review+Guidelines+and+approved+practices)

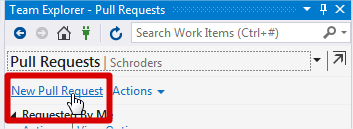
* Before creating pull request (or before adding to a PR) conduct peer review to eliminate obvious misunderstandings
* When doing **asserts** in unit tests make comments inside Asserts where it is desirable (not in every assert)
  + **DO** put messages inside asserts => Assert.IsGreaterThan(5, someVariable, "comment");
  + **DON'T** put messages in comments outside of assert => Assert.IsGreaterThan(5, someVariable); //comment
* Use consistently either MSTest **Asserts** or **FluentAssertion** per one class. **Do not mix** two libraries in one test class
* Comment every public method, interface or property if it's not guessable
  + **DO** comment manually and make useful comments
  + **DON'T** auto-generate comments and don't comment obvious things that can be understood only by reading eg. method name
* **Private fields** should be prefixed with underscore
  + **DO**\_someField.Something();
  + **DON'T**this.\_someField.Something(); or this.someField.Something(); or someField.Something();
* **Write unit tests** wherever it makes sense, which is almost all logic
* The function does not crash the console (using normal parameters) or Service Fabric   
  [this check can be performed on an owned developer machine or the author quickly runs the function on his machine]

# [Create a pull-request to 'master'](https://confluence.schroders.com/pages/viewpage.action?pageId=34244040)

This  article briefly describes how the master branch should be merged with a development branch.

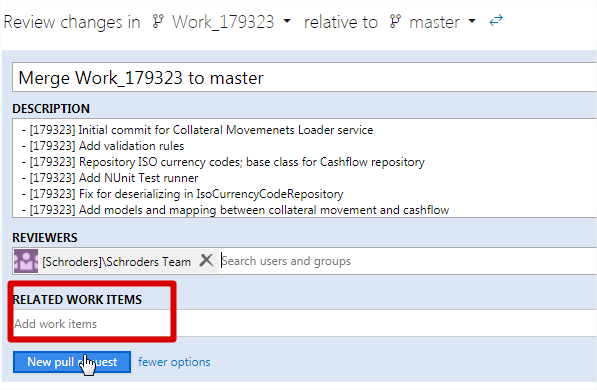
## Create a pull request in Visual Studio





→ A web page should open

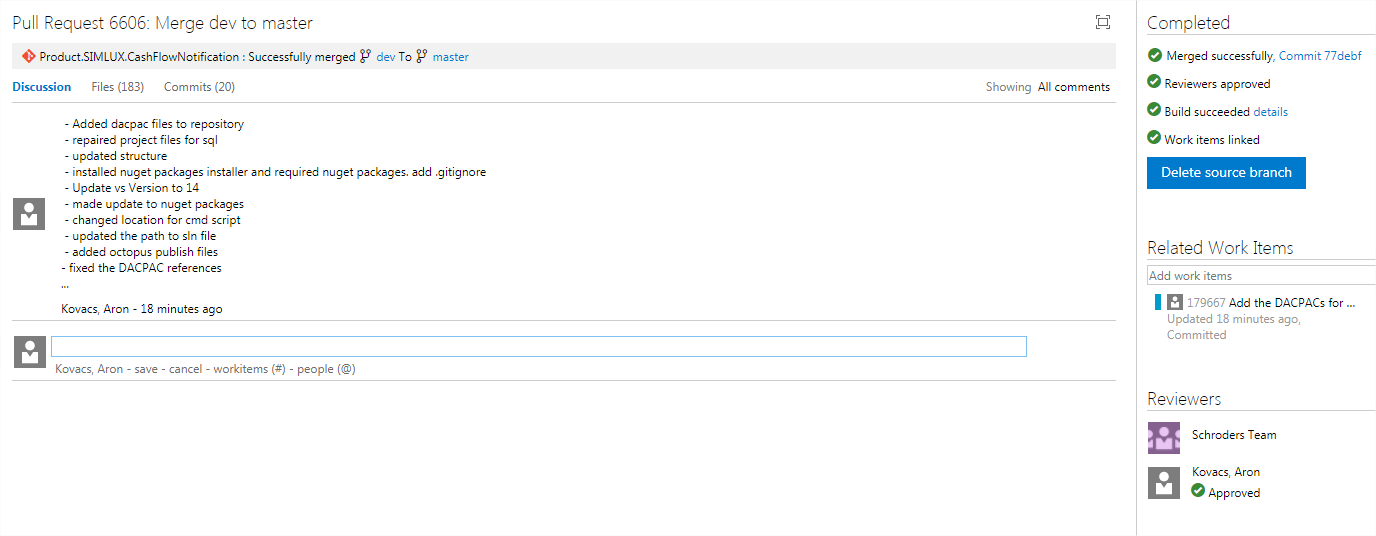
## Review pull-request details



* Verify the description
* Verify the related work items
* Review the file list

## Approve a pull-request

Once the pull-request is created, it needs to be approved (by a reviewer). The page looks similar to the one below:



Please note:

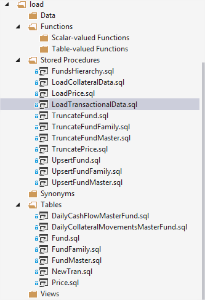
* The request can be abandoned or completed
* A build is automatically kicked-off. Failure or success is indicated. If there is a problem with the build definition, then the build can be restarted
* The reviewer has to approve the request
* Upon completion, the branch can be automatically deleted if you are owning the branch. Otherwise the owner has to be notified.

# [Database Changes (DACPAC)](https://confluence.schroders.com/pages/viewpage.action?pageId=42505124)

This page describes the patterns and practices to be applied with the DACPACs under our responsibility.

## Tables, Functions, Stored Procedures

Tables, functions and stored procedures need to be stored in a folder indicating the schema to which those sql object below to:

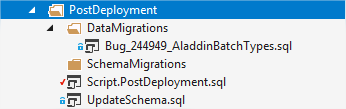


* Stored procedures and functions do not need to be idempotent in this case
* Constraints applied on tables must not break any existing instance (new unique indexes, etc), for a case where a **pre-deployment**migration is required in order to be able to install an index, see 'Schema Migrations'

## Schema and Data Migrations

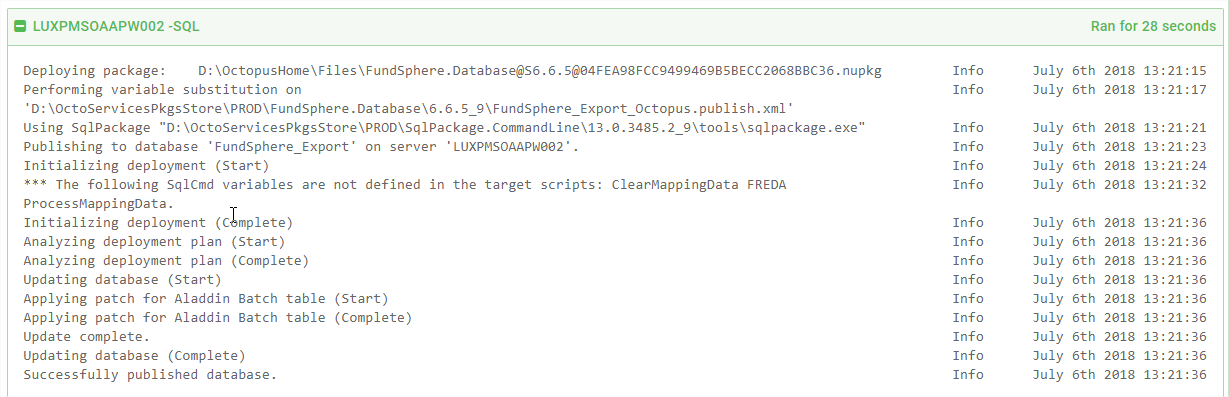
Schema and data migration scripts...

* ...**are always idempotent**
* ...must identify the data to migrate clearly
* ...may require data changes as well, which then can be kept in the schema migration script
* ...must be fencing their activity in print statements (so that Octopus can be a bit more clear in case of an error
* ...must be in dedicated files following the branch name and a short(!) hint on the migrated entities
* ...which must be executed in a strict order, need to be referenced by the UpdateSchema.sql and have build action 'none'



**Example Schema/Data Migration Script**

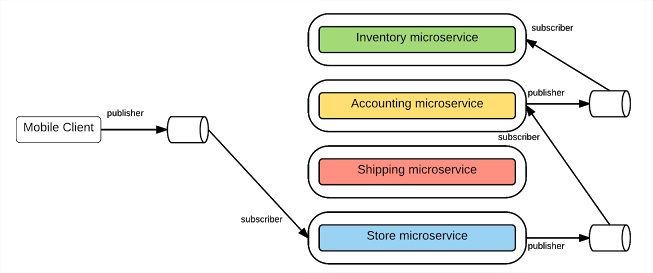
|  |
| --- |
| /\*  Post-Deployment Script Template  --------------------------------------------------------------------------------------   This file contains SQL statements that will be appended to the build script.   Use SQLCMD syntax to include a file in the post-deployment script.   Example:      :r .\myfile.sql   Use SQLCMD syntax to reference a variable in the post-deployment script.   Example:      :setvar TableName MyTable                 SELECT \* FROM [$(TableName)]  --------------------------------------------------------------------------------------  \*/    print 'Applying patch for Aladdin Batch table (Start)'    update flows.AladdinBatch set          BatchType = CASE WHEN CharIndex('cancel', FileName) >0 THEN 'CANCELCASHFLOW'                           ELSE 'CASHFLOW' END,          FileTimeStamp = CreatedOn,          StatusCode = 'DISTRIBUTED' -- This is ok, only old batches do not have the batchtype set (because it was not in use).      where BatchType is null    print 'Applying patch for Aladdin Batch table (Complete)' |



Octopus screenshot of deployment containing the print statements  
  
 After all environments have been migrated with the DACPAC, the schema migration may be removed (it would be no-op on all environments)

# [Interaction with Messages Queues](https://confluence.schroders.com/display/GCFN/Interaction+with+Messages+Queues)

Messages queues are middle ware which cover communication scenarios in which the client is not interested in a response of the service provider. An example on how micro-services can be attached to message queues is the following:



The interaction with a message queue happens through two roles: the consumer and the provider.

* [The Consumer](https://confluence.schroders.com/display/GCFN/Interaction+with+Messages+Queues#InteractionwithMessagesQueues-TheConsumer)
* [The Provider](https://confluence.schroders.com/display/GCFN/Interaction+with+Messages+Queues#InteractionwithMessagesQueues-TheProvider)
* [The DO's and DON'Ts](https://confluence.schroders.com/display/GCFN/Interaction+with+Messages+Queues#InteractionwithMessagesQueues-TheDO'sandDON'Ts)

## The Consumer

A class which is taking messages from Messages Queues, such as KAFKA, is named Consumers. This is because they are supposed to consume a message created by the producer.

* a consumer uses a broker to subscribe to a topic
* a consumer always validates the incoming string message prior parsing it. This validation has to use a well known methodology (either XSD or OpenAPI, see [ReDoc](https://github.com/Rebilly/ReDoc)). If validation fails, the default behavior is
  + to log an error
  + stop further processing
  + commit the transaction to the queue
* a consumer is accessed through a contract (e.g. an interface). This aims for test ability; in a unit test environment consumers will listen to mocks so that the business logic of the application can be tested.

## The Provider

A class which is putting messages to a Message Queue, such as KAFKA, is named Producer. This is because it is producing data for other consumers. A consumer typically:

* Produces a message which contains all relevant information for consumers. They should have to call back the producer's system for additional information. If this is not respected, messages cannot be replayed as information not in being part of the message may have change in the time between making the replayed message an invalid request.
* a producer is accessed through a contract (e.g. an interface). This aims for test ability; in a unit test environment prodivers will be mocked with dummies so that other tests do not require the deployment of a queue.

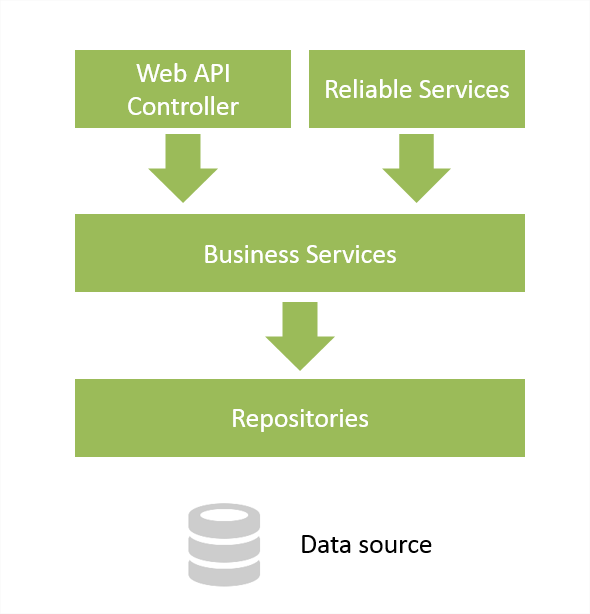
## The DO's and DON'Ts

# [Repositories: Interaction with Data Sources](https://confluence.schroders.com/display/GCFN/Repositories%3A+Interaction+with+Data+Sources)

Classes which read/write/manipulate data from/to a data source (a database, file, etc...) are named repositories. The point on calling this class a repository is that it needs to be understood agnostic on the data source. A fund repository could use a file, a database, a web service as its data source, or even multiple data sources at the same time. Typically,

* [Hierarchy](https://confluence.schroders.com/display/GCFN/Repositories%3A+Interaction+with+Data+Sources#Repositories:InteractionwithDataSources-Hierarchy)
* [DO's and DON'Ts](https://confluence.schroders.com/display/GCFN/Repositories%3A+Interaction+with+Data+Sources#Repositories:InteractionwithDataSources-DO'sandDON'Ts)
* [Common Base Types](https://confluence.schroders.com/display/GCFN/Repositories%3A+Interaction+with+Data+Sources#Repositories:InteractionwithDataSources-CommonBaseTypes)
* [Entity Design Guidelines](https://confluence.schroders.com/display/GCFN/Repositories%3A+Interaction+with+Data+Sources#Repositories:InteractionwithDataSources-EntityDesignGuidelines)
  + [Entity Property Name Guidelines](https://confluence.schroders.com/display/GCFN/Repositories%3A+Interaction+with+Data+Sources#Repositories:InteractionwithDataSources-EntityPropertyNameGuidelines)
  + [Entity Codification/ Enumeration Guidelines](https://confluence.schroders.com/display/GCFN/Repositories%3A+Interaction+with+Data+Sources#Repositories:InteractionwithDataSources-EntityCodification/EnumerationGuidelines)

## Hierarchy



## DO's and DON'Ts

| **A repository is typically** | **A repository is not** |
| --- | --- |
| The repository will only read/ write/ search data | The repository will not adapt/ mutate/ calculate additional data |
| A repository is only responsible for a single one type of object (can be entity, poco, domain object) | The repository does not represent the entry point for plentiful different objects |
| The repository object is strongly typed | The repository is a dynamic object or a dictionary type |
| The repository can return objects which are payloads of the root object |  |
| **All** repositories implement a single standard contract: Items  IEntityRepository   Not all methods need to be implemented by all repositories; i.e. NotSupportedException is ok. | The repository is having unique methods to CRUD data (more hard to unit test) |
| The repository can use other helper classes/ APIs to perform its work: Entity Framework, CSV Helper, ADO.NET. |  |
| The repository validates data to be written (if applicable). Examples for data validation: field lengths, nulls | The repository is applying business object validation |
| The repository can sanitize data, i.e. the repository is allowed to eliminate the bias between <null> and "" for fields  This aims for cases when the data source is not under our control, but the bias could cause problems |  |
| The repository does not implement business functions |  |
| The repository completely encapsulates its data source. For instance not allowed to throw SqlExceptions of any kind. If these may occur (index violation, the repository needs to catch this exception and throw its own exception). Catching 'Exception' of course is forbidden | The repository is not throwing SqlExceptions, FileSystemExceptions, WebExceptions by design |
| The repository is not using any business services. It may however use web services. | The repository is not using one or more business services, is not using other repositories |
| The repository is supporting transactions. Please note that the scope of the transaction may be limited (Sql or Reliable Storage or both?) and a transaction service is responsible for the scope of it). | The repository is committing a transaction |

## Common Base Types

* **EntityRepository**: a repository which is connected to an Entity **Framework** Data Context
* **AdoRepository**: a repository which is connected to an ADO.NET connection
* **CsvRepository**: a repository which is connected to a CSV stream(!)

## Entity Design Guidelines

* Entities can represent a single table or a combination of those
* Entities can host many-to-one and one-to-many projections
* Entities should not be serializable (infine loops!)

### Entity Property Name Guidelines

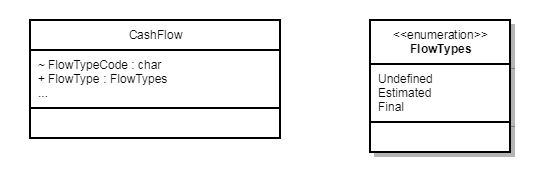
* Entity Property Names must be without typos and meaningful
* Entity Property Names must have a summary which is derived or equal to the mapping specification.
* Entity Property Names must not contain typos or abbreviations originating from the underlying column

Entity Property Names must be decorated with cardinalities and data limitations (see [Data Annotation Attributes](http://www.entityframeworktutorial.net/code-first/required-attribute-dataannotations-in-code-first.aspx))

### Entity Codification/ Enumeration Guidelines

In many cases, a table may contain a status or type column and -at the same time- these columns have often a big impact in code as rules are changing with their value. In order to keep the code more readable, it is expected to eliminate the useless complexity which comes from the string or character representation.

This is achieved by wrapping these values into enumerations. See the following example:



If column 'FlowType' can have character 'E' which represents estimated.

* + The entity should refer to the column as 'FlowType**Char**' or 'FlowType**Code**' as **internal** char
  + The entity should refer to the column as 'FlowType' as a **public** enumeration 'FlowType**s**'.
  + The property setters populate always a single backing field, in most cases the enum is the better choice. The adaptaion is done by an adapter (hugh!)
  + The enumeration must have 'Undefined' as its default value
  + The enumeration property must not be nullable

Code Examples:

**Cash Flow Entity**

**CashFlow Entity**

|  |
| --- |
| public class CashFlow     {         private FlowTypes flowType;           /// <summary>         /// Gets or sets the flow type code.         /// </summary>         internal char FlowTypeCode { get { return FlowTypeAdapter.ToCode(flowType); } set { flowType = FlowTypeAdapter.From(value); } }           /// <summary>         /// Gets or sets the flow type.         /// </summary>         [NotMapped]         public FlowTypes FlowType { get { return flowType; } set { flowType = value; } }     } |

**Flow Type Adapter**

**Flow Type Adapter** Expand source

**Flow usage Examples**

**Flow Usage Examples**

|  |
| --- |
| private void FlowExamples ()  {              CashFlow flow = new CashFlow();                switch (flow.FlowType)              {                  case FlowTypes.Undefined:                      break;                  case FlowTypes.Estimated:                      DoOtherStuff();                      break;                  case FlowTypes.Final:                      DoOtherStuff();                      break;                  default:                      throw new NotSupportedException();              }  } |

**Adapter Usage Examples:**

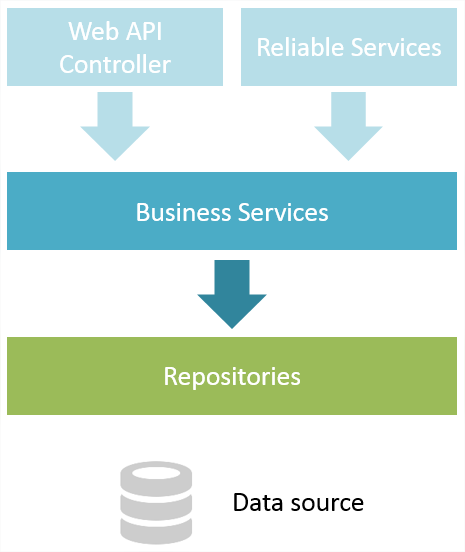
**Adapter Usage Examples**

|  |
| --- |
| private void Examples()  {              FlowTypes t1 = FlowTypeAdapter.From('e');              FlowTypes t2 = FlowTypeAdapter.From('E');              FlowTypes t3 = FlowTypeAdapter.From("e");              FlowTypes t4 = FlowTypeAdapter.From("E");              FlowTypes t5 = FlowTypeAdapter.From("EstMated");  } |

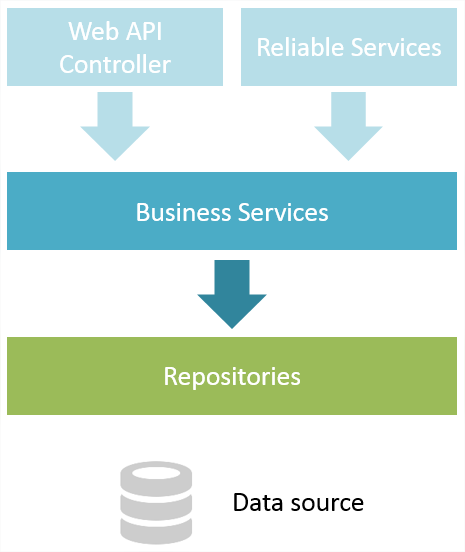
# [Services: Overview](https://confluence.schroders.com/display/GCFN/Services%3A+Overview)

Services are a generic term for a certain encapsulated functionality. In this project, we focus on the follow service types, which are explained further:

* Business Services, see [Services: Business Services](https://confluence.schroders.com/display/GCFN/Services%3A+Business+Services)
* Web Service Client Services (urgh!), see [Services: Service Clients](https://confluence.schroders.com/display/GCFN/Services%3A+Service+Clients)
* Web Services, REST and Reliable Services, see [Services: web-services and micro services](https://confluence.schroders.com/display/GCFN/Services%3A+web-services+and+micro+services), and [REST Guidelines](https://confluence.schroders.com/display/GCFN/REST+Guidelines)



# [Services: Business Services](https://confluence.schroders.com/display/GCFN/Services%3A+Business+Services)



## Business Services

Business Services are implementations of a tailor made service contact fullfilling some purpose, as with the AOSP principle, business services are most of the time one of the two kinds

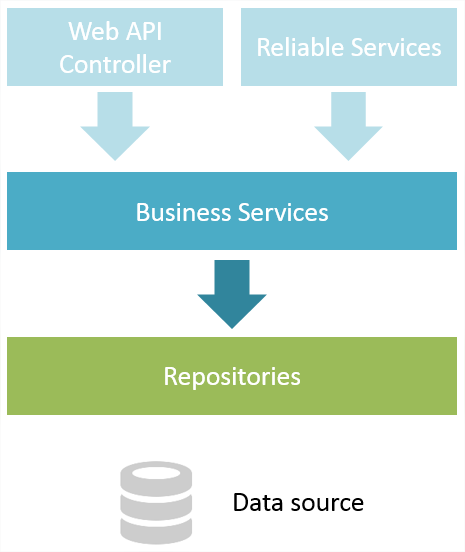
1. Business Services which actually perform stuff (calculations)
2. Business Services which orchestrate other (business) services, repositories in order to achieve some goal.

| **DO** | **DON'T** |
| --- | --- |
| Validate input data (under business characteristics) |  |
| Perform any operation which doesn't leave the tier | Load/ save data **and** do other stuff |
| Use repositories to get/ put data |  |
| Use other services |  |
| Use other web services |  |
| All the logic is unit tested | Avoid to test logic which is not part of the service, use mocks in tests if necessary |
| Can be mocked in unit tests |  |

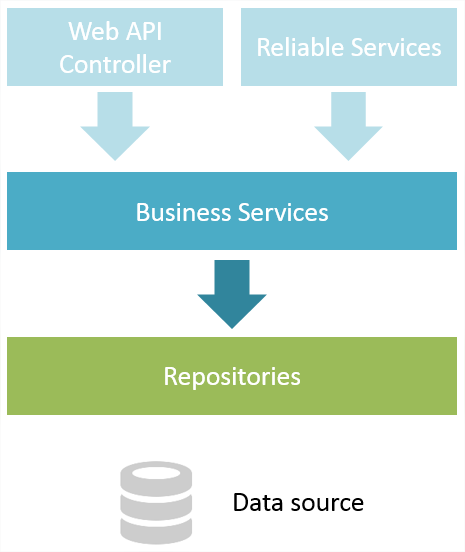
# [Services: Service Clients](https://confluence.schroders.com/display/GCFN/Services%3A+Service+Clients)

Service clients are client components of external web-services. There are two common types of Service Clients:

1. REST Service Clients
2. Reliable Service Clients



# [Services: web-services and micro services](https://confluence.schroders.com/display/GCFN/Services%3A+web-services+and+micro+services)



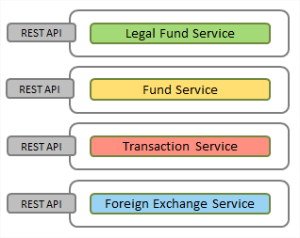
Parts of this chapter are an abstract from a good design guideline article, which can be read in complete [here](https://medium.com/@WSO2/guidelines-for-designing-microservices-71ee1997776c).

* [Problem Statement](https://confluence.schroders.com/display/GCFN/Services%3A+web-services+and+micro+services#Services:web-servicesandmicroservices-ProblemStatement)
* [Service Fabric](https://confluence.schroders.com/display/GCFN/Services%3A+web-services+and+micro+services#Services:web-servicesandmicroservices-ServiceFabric)
* [Service Remoting](https://confluence.schroders.com/display/GCFN/Services%3A+web-services+and+micro+services#Services:web-servicesandmicroservices-ServiceRemoting)
* [Service Design Pattern](https://confluence.schroders.com/display/GCFN/Services%3A+web-services+and+micro+services#Services:web-servicesandmicroservices-ServiceDesignPattern)
* [Naming Conventions](https://confluence.schroders.com/display/GCFN/Services%3A+web-services+and+micro+services#Services:web-servicesandmicroservices-NamingConventions)
* [The DO's and DON'Ts](https://confluence.schroders.com/display/GCFN/Services%3A+web-services+and+micro+services#Services:web-servicesandmicroservices-TheDO'sandDON'Ts)

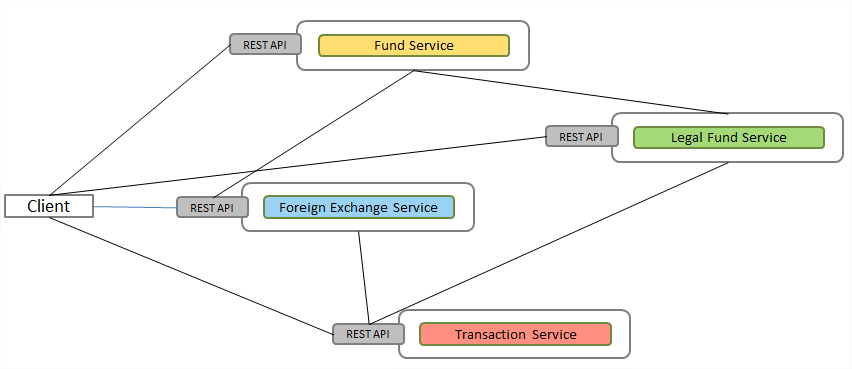
## Problem Statement

Web services are a pattern which allow clients to disconnect from the source of the data. While traditionally data exchange between applications may have happened through database mirrors or cross database access, which has proven to be difficult to manage and maintain, web services are the preferred way today. As an evolutionary step to the SOA, a service oriented architecture, the micro service architecture is useful to be known. Services in SOA have the tendency to become fat and complex to use as they need to cover all use cases and features for all clients.

Micro-services then, although the term 'micro' is indeed a bit misleading, try to stick the service capability to the right scope. That said, they are not necessarily small, but it should be tried to make them as small as possible (slice and dice). The exposure of those services is supposed to happen through REST which leads to an architecture like the following:



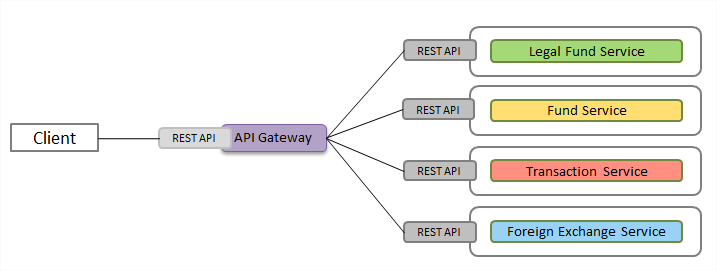
Without further governance, SOA or micro services the integration requirements will likely lead to a structure like the following (the web):



This leads to several problems:

* The services are becoming interdependent, workflows are difficult to reconstruct (traces, logs, telemetry)
* Deployment becomes complex due to dependencies of the dependencies
* Configuration of the microservice interactions can become a challenge
* The services will become more and more complex due to the needs of a single client, the risk for procedural programming increases
* Performance can be difficult/ impossible to predict and maintain as a call initiates practically an unknown number of network based service calls

In order to overcome most of the issues, an API Gateway can be introduced which shields the (micro) services from the clients (this takes away most of the problems above):



## Service Fabric

Azure Service Fabric is a distributed system which makes it easy to package, deploy, monitor and host web and micro services. Complex infrastructure and configuration problems can be avoided by hosting services in the local azure container. Extensible documentation is available on [MSDN](https://docs.microsoft.com/en-us/azure/service-fabric/). In essence Service Fabric has been selected because it allows:

* Scaleable **stateful and stateless** services: service fabric manages the nodes which host services with almost simultaneous fail-over (so named reliable services)
* Service Remoting allows not to bother about the actual URLs of a particular service at all; instead services are references through the fabric protocol: ''

## https://confluence.schroders.com/download/attachments/29764822/image2017-12-29_9-44-24.png?version=1&modificationDate=1514537064310&api=v2

Service Fabric can host three types of services:

## https://confluence.schroders.com/download/attachments/29764822/image2017-12-29_11-3-24.png?version=1&modificationDate=1514541804163&api=v2

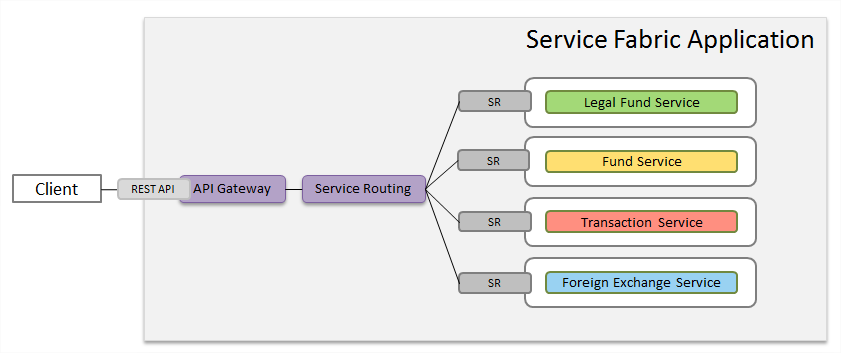
Service Fabric offers a built-in stateful service which supports transactions over multiple (micro) services. Bear in mind that our primary data source remains SqlServer so the Reliable Storage must not be used for production data, but only for state representation.

## Service Remoting

As mentioned above, an API gateway is recommended in order to streamline incoming client requests.The recommended protocol on API gateways is REST. Internally however, we need a more efficient and fast approach -especially because those services will be hosted on many nodes and likely interact very often. Service Remoting is a term which for us refers to Service Fabric Service Remoting, this is the fastest and easiest way to integrate web/ micro services in the API gateway:

* Automatic service address resolution
* Connection management **and retries**
* Error handling
* Strong typed
* Fast

## Service Design Pattern



## Naming Conventions

* Use the 'WebService' suffix for contracts of Remoted Services, i.e. FundWebService

## The DO's and DON'Ts

| **DO** | **DO NOT** |
| --- | --- |
| Define the service contract in such a way that batch processing is possible | Use web-services in loops (for each...) |
| Use the suffix 'WebService' in the contract name to flag the performance implications |  |
| Use the suffix 'ReliableService' in the contract name to distinguish it immediately from business services (in 'Fabric' namespace) |  |
| Call a business service to perform the activity | Implement logic directly on the reliable service/ controller |
| Use standard REST routes, for client APIs, see [REST Guidelines](https://confluence.schroders.com/display/GCFN/REST+Guidelines) | Invent own/ custom routing logic |
| Use Reliable Storage for temporary and state data | Use Reliable Storage to store production data |
| Reliable Services are **async implementations** |  |
| REST Services are **sync implementations** |  |
| A unit test only verifies that the service call is as expected (all parameters, etc) | Test functionality of the used services, the web service class is only supposed to forward the call to the business service |

# [Services: Service Fabric Health](https://confluence.schroders.com/display/GCFN/Services%3A+Service+Fabric+Health)

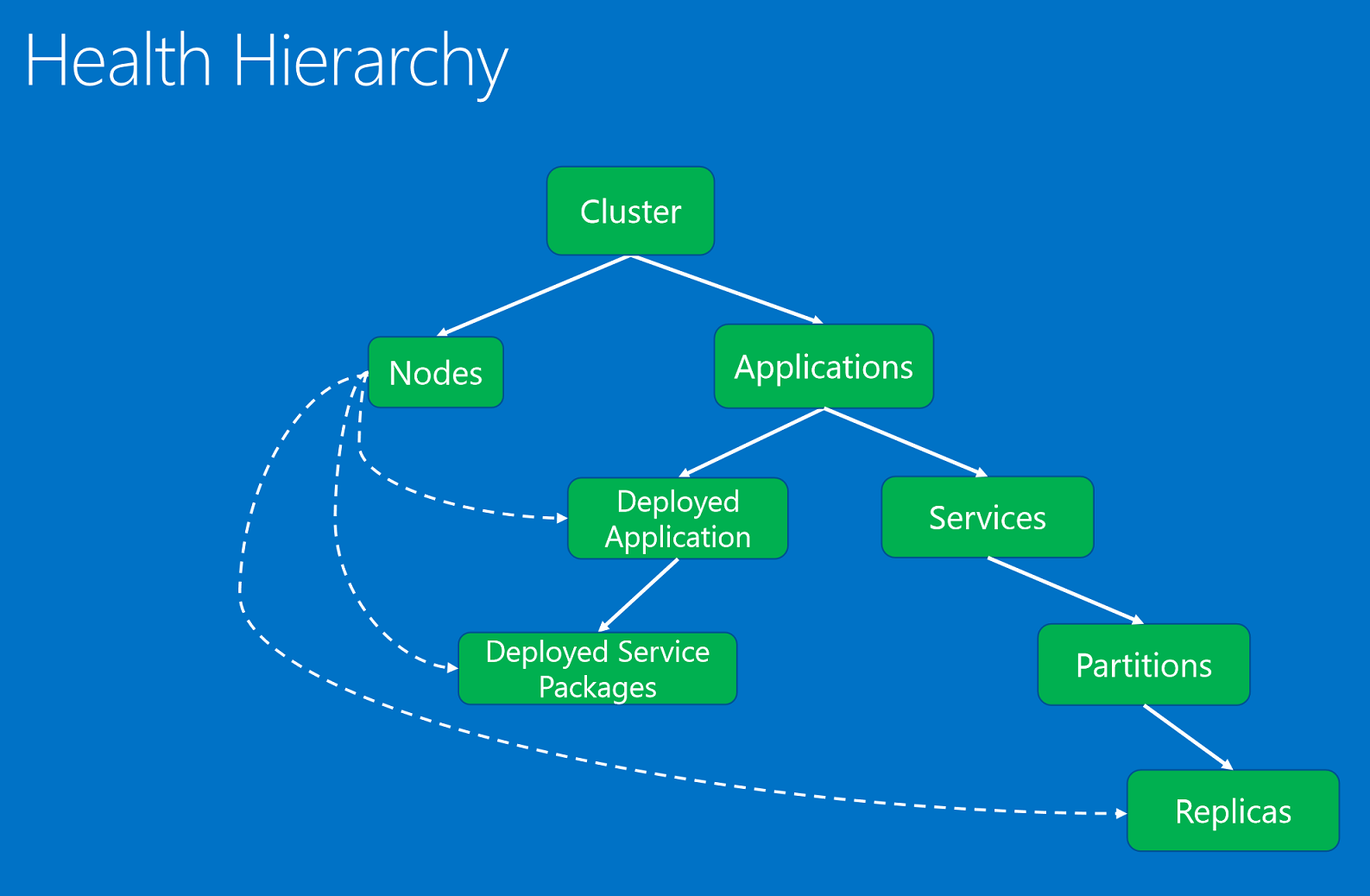
* [Health store](https://confluence.schroders.com/display/GCFN/Services%3A+Service+Fabric+Health#Services:ServiceFabricHealth-Healthstore)
* [Health entities and hierarchy](https://confluence.schroders.com/display/GCFN/Services%3A+Service+Fabric+Health#Services:ServiceFabricHealth-Healthentitiesandhierarchy)
* [Health states](https://confluence.schroders.com/display/GCFN/Services%3A+Service+Fabric+Health#Services:ServiceFabricHealth-Healthstates)
* [Health policies](https://confluence.schroders.com/display/GCFN/Services%3A+Service+Fabric+Health#Services:ServiceFabricHealth-Healthpolicies)
  + [Cluster health policy](https://confluence.schroders.com/display/GCFN/Services%3A+Service+Fabric+Health#Services:ServiceFabricHealth-Clusterhealthpolicy)
  + [Application health policy](https://confluence.schroders.com/display/GCFN/Services%3A+Service+Fabric+Health#Services:ServiceFabricHealth-Applicationhealthpolicy)
  + [Service type health policy](https://confluence.schroders.com/display/GCFN/Services%3A+Service+Fabric+Health#Services:ServiceFabricHealth-Servicetypehealthpolicy)
* [Health evaluation](https://confluence.schroders.com/display/GCFN/Services%3A+Service+Fabric+Health#Services:ServiceFabricHealth-Healthevaluation)
* [Custom health reports](https://confluence.schroders.com/display/GCFN/Services%3A+Service+Fabric+Health#Services:ServiceFabricHealth-Customhealthreports)
* [Example](https://confluence.schroders.com/display/GCFN/Services%3A+Service+Fabric+Health#Services:ServiceFabricHealth-Example)
* [Reference](https://confluence.schroders.com/display/GCFN/Services%3A+Service+Fabric+Health#Services:ServiceFabricHealth-Reference)

# Health store

The health store keeps health-related information about entities in the cluster for easy retrieval and evaluation.

# Health entities and hierarchy

The health entities are organized in a logical hierarchy (parent-child relationships) that captures interactions and dependencies among different entities. The health store automatically builds health entities and hierarchy based on reports received from Service Fabric components.



# Health states

* **OK.** The entity is healthy.
* **Warning.** The entity has some issues, but it can still function correctly.
* **Error.** The entity is unhealthy. Action should be taken to fix the state of the entity, because it can't function properly.
* **Unknown.** The entity doesn't exist in the health store. An entity is not in store because health reports have not yet been processed or the entity has been cleaned up after deletion.

# Health policies

The health store applies health policies to determine whether an entity is healthy based on its reports and its children. By default, Service Fabric applies strict rules (everything must be healthy) for the parent-child hierarchical relationship.

## Cluster health policy

* ConsiderWarningAsError
* MaxPercentUnhealthyApplications
* MaxPercentUnhealthyNodes
* ApplicationTypeHealthPolicyMap

## Application health policy

* ConsiderWarningAsError
* MaxPercentUnhealthyDeployedApplications
* DefaultServiceTypeHealthPolicy
* ServiceTypeHealthPolicyMap

## Service type health policy

* MaxPercentUnhealthyPartitionsPerService
* MaxPercentUnhealthyReplicasPerPartition
* MaxPercentUnhealthyServices

# Health evaluation

Users and automated services can evaluate health for any entity at any time. To evaluate an entity's health, the health store aggregates all health reports on the entity and evaluates all its children (when applicable). The health aggregation algorithm uses health policies that specify how to evaluate health reports and how to aggregate child health states (when applicable).

# Custom health reports

If Service Fabric services do not have admin access to the cluster, you can report health on entities from the current context through **Partition**or **CodePackageActivationContext**.

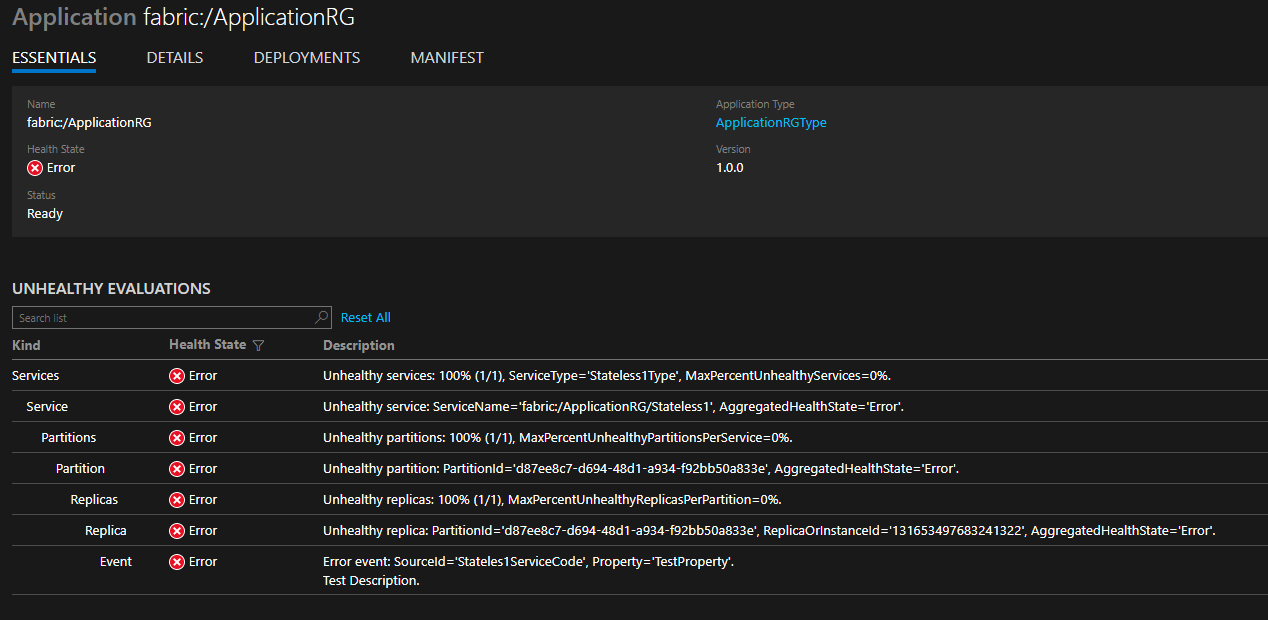
* For stateless services, use **IStatelessServicePartition.ReportInstanceHealth** to report on the current service instance.
* For stateful services, use **IStatefulServicePartition.ReportReplicaHealth** to report on current replica.
* Use **IServicePartition.ReportPartitionHealth** to report on the current partition entity.
* Use **CodePackageActivationContext.ReportApplicationHealth** to report on current application.
* Use **CodePackageActivationContext.ReportDeployedApplicationHealth** to report on the current application deployed on the current node.
* Use **CodePackageActivationContext.ReportDeployedServicePackageHealth** to report on a service package for the application deployed on the current node.

You can specify **HealthReportSendOptions**when sending reports through **Partition**and **CodePackageActivationContext**health APIs. If you have critical reports that must be sent as soon as possible, use **HealthReportSendOptions**with **Immediate**true. Immediate reports bypass the batching interval of the internal health client.

# Example

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | //example for stateless service  var health = new HealthInformation("Stateles1ServiceCode", "TestProperty", HealthState.Error)  {      Description = "Test Description."  };  var options = new HealthReportSendOptions { Immediate = true };  this.Partition.ReportInstanceHealth(health, options); |

Result of the example code:



# Reference

* <https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-health-introduction>
* <https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-diagnostics-how-to-report-and-check-service-health>
* <https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-report-health>

# [Services: Service Fabric reliable services](https://confluence.schroders.com/display/GCFN/Services%3A+Service+Fabric+reliable+services)

* Created by Unknown User (jeskedo), last modified on [Jun 11, 2019](https://confluence.schroders.com/pages/diffpagesbyversion.action?pageId=74034510&selectedPageVersions=1&selectedPageVersions=2)

[Go to start of metadata](https://confluence.schroders.com/display/GCFN/Services%3A+Service+Fabric+reliable+services#page-metadata-start)

* [General rules](https://confluence.schroders.com/display/GCFN/Services%3A+Service+Fabric+reliable+services#Services:ServiceFabricreliableservices-Generalrules)
* [FSI rules](https://confluence.schroders.com/display/GCFN/Services%3A+Service+Fabric+reliable+services#Services:ServiceFabricreliableservices-FSIrules)

# ****General rules****

* Method names should have **distinct names** (method override is not supported)
* Method arguments **should not have default parameters**
* Service contracts should inherit from **IService** interface

# ****FSI rules****

* To enable interception **all public methods should be virtual**
* File FundSphere.ServiceFabricCommon\Logging\ServiceEventSource.cs should be added as link to configuration folder of each service and additionally there should be partial class with the same name and relevant attribute:   
  [EventSource(Name = "FSI.Orchestrator")]
* Every service should be decorated with **ServiceAttribute** with service type name, type of the catalog and TTL for service health: [Service("FSI.Load.ExchangeRatesType", typeof(LoadExchangeRateCatalog), 125)]
* Service should derive from **FsiStatelessService** for stateless services or **FsiStatefullService** for statefull services
* Service contracts should be decorated with **ServiceClient** attribute

[Setting up AutoMapper](https://confluence.schroders.com/display/GCFN/Using+AutoMapper#UsingAutoMapper-SettingupAutoMapper)

* [Basic ways of mapping objects](https://confluence.schroders.com/display/GCFN/Using+AutoMapper#UsingAutoMapper-Basicwaysofmappingobjects)
  + [Automatic mapping](https://confluence.schroders.com/display/GCFN/Using+AutoMapper#UsingAutoMapper-Automaticmapping)
  + [Flattening mapping](https://confluence.schroders.com/display/GCFN/Using+AutoMapper#UsingAutoMapper-Flatteningmapping)
  + [Reverse mapping and Unflattening](https://confluence.schroders.com/display/GCFN/Using+AutoMapper#UsingAutoMapper-ReversemappingandUnflattening)
  + [Attribute mapping](https://confluence.schroders.com/display/GCFN/Using+AutoMapper#UsingAutoMapper-Attributemapping)
  + [Projection mapping](https://confluence.schroders.com/display/GCFN/Using+AutoMapper#UsingAutoMapper-Projectionmapping)
  + [Collection mapping](https://confluence.schroders.com/display/GCFN/Using+AutoMapper#UsingAutoMapper-Collectionmapping)
  + [Open generics mapping](https://confluence.schroders.com/display/GCFN/Using+AutoMapper#UsingAutoMapper-Opengenericsmapping)
* [Profile Instances](https://confluence.schroders.com/display/GCFN/Using+AutoMapper#UsingAutoMapper-ProfileInstances)
  + [Automatically scanning for profiles](https://confluence.schroders.com/display/GCFN/Using+AutoMapper#UsingAutoMapper-Automaticallyscanningforprofiles)
* [Automapper configuration validation](https://confluence.schroders.com/display/GCFN/Using+AutoMapper#UsingAutoMapper-Automapperconfigurationvalidation)
* [Working examples](https://confluence.schroders.com/display/GCFN/Using+AutoMapper#UsingAutoMapper-Workingexamples)
* [Reference](https://confluence.schroders.com/display/GCFN/Using+AutoMapper#UsingAutoMapper-Reference)

# Setting up AutoMapper

We can use Static or Instance version of AutoMapper API.  
All examples from LuxGlobalTACashflowNotifications\Examples\Service Fabric Schroders Demo\SIM.TA.FundService.Tests\AutoMapper\ repository are created using Instance API.

First we need to define mappings in MapperConfiguration.

|  |  |
| --- | --- |
| 1  2  3  4  5 | var config = new MapperConfiguration(cfg =>  {      cfg.CreateMap<SourceType1, DestinationType1>();      cfg.CreateMap<SourceType2, DestinationType2();  }); |

Using config object we can create IMapper instance that can later be used in DI Container.

|  |  |
| --- | --- |
| 1 | IMapper mapper = config.CreateMapper(); |

# Basic ways of mapping objects

## Automatic mapping

Source member with example name "FirstName" will be automatically mapped to Destination member with name "FirstName".

**Example**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | public class Source  {      public string FirstName { get; set; }  }    public class Destination  {      public string FirstName { get; set; }  }    var config = new MapperConfiguration(cfg =>  {      cfg.CreateMap<Source, Destination>();  });    var mapper = config.CreateMapper();    var src = new Source { FirstName = "TestName" };  var dest = mapper.Map<Destination>(src); |

## Flattening mapping

AutoMapper splits the destination member name into individual words by PascalCase conventions.

**Example**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | public class Customer  {      public string Name { get; set; }  }    public class Source  {      public Customer Customer { get; set; }  }    public class Destination  {      public string CustomerName { get; set; }  }    var config = new MapperConfiguration(cfg =>  {      cfg.CreateMap<Source, Destination>();  });    var mapper = config.CreateMapper();    var src = new Source { Customer = new Customer { Name = "TestName" } };  var dest = mapper.Map<Destination>(src); |

## Reverse mapping and Unflattening

We can map in both directions.

**Example**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | var config = new MapperConfiguration(cfg =>  {      cfg.CreateMap<Source, Destination>().ReverseMap();  });    var mapper = config.CreateMapper();    var src = new Source { FirstName = "TestName" };  var dest = mapper.Map<Destination>(src);      dest.FirstName = "NewName";  mapper.Map(dest, src);  // src.FirstName should be "NewName" |

## Attribute mapping

We can use attribute directly on member.

**Example**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | public class Source  {      [MapTo("PropertyTwo")]      public string PropertyOne { get; set; }  }    public class Destination  {      public string PropertyTwo { get; set; }  }    var config = new MapperConfiguration(cfg =>  {      cfg.CreateMap<Source, Destination>();  });    var mapper = config.CreateMapper();    var src = new Source { PropertyOne = "TestValue" };  var dest = mapper.Map<Destination>(src);  // dest.PropertyTwo should be "TestValue" |

## Projection mapping

We can specify custom member mapping definitions.

**Example**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | public class Source  {      public DateTime Date { get; set; }  }    public class Destination  {      public int EventHour { get; set; }      public int EventMinute { get; set; }  }    var config = new MapperConfiguration(cfg =>  {      cfg.CreateMap<Source, Destination>()          .ForMember(dest => dest.EventHour, opt => opt.MapFrom(src => src.Date.Hour))          .ForMember(dest => dest.EventMinute, opt => opt.MapFrom(src => src.Date.Minute));  });    var mapper = config.CreateMapper();    var src = new Source { Date = new DateTime(2018, 2, 21, 13, 10, 0) };  var dest = mapper.Map<Destination>(src); |

## Collection mapping

**Example**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | var config = new MapperConfiguration(cfg =>  {      cfg.CreateMap<Source, Destination>();  });    var mapper = config.CreateMapper();    var itemsSrc = new[] { new Source { Name = "Name1" }, new Source { Name = "Name2" } };  var listDest = mapper.Map<Source[], List<Destination>>(itemsSrc);  var arrayDest = mapper.Map<Source[], Destination[]>(itemsSrc); |

## Open generics mapping

**Example**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | public class Source<T>  {      public T Value { get; set; }  }    public class Destination<T>  {      public T Value { get; set; }  }    var config = new MapperConfiguration(cfg =>  {      cfg.CreateMap(typeof(Source<>), typeof(Destination<>));  });    var mapper = config.CreateMapper();      var src = new Source<int> { Value = 11 };  var dest = mapper.Map<Destination<int>>(src); |

# Profile Instances

Profiles are usefull when we want to organize our mappings. To create custom profile we need to inherit AutoMapper Profile abstract class.  
Profiles can be used in applications with more than one layer e.g. UI/Domain/DB. With one configuration class we would have to reference DB project in UI layer, otherwise we couldn't map our objects.

**Example**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | // we save this profile in ui project  public class UiToDomainProfile : Profile  {      public UiToDomainProfile()      {          CreateMap<CarViewModel, CarDomain>();      }  }    // we save this profile in domain project  public class DomainToDbProfile : Profile  {      public DomainToDbProfile()      {          CreateMap<CarDomain, CarEntity>();      }  }    var config = new MapperConfig(cfg =>  {      //we can manually add profiles to mapper configuration      cfg.AddProfile(new UiToDomainProfile());      cfg.AddProfile(new DomainToDbProfile());  });    var mapper = config.CreateMapper(); |

## Automatically scanning for profiles

Scan for all profiles in an assembly

|  |  |
| --- | --- |
| 1  2  3 | var config = new MapperConfiguration(cfg => {      cfg.AddProfiles(myAssembly);  }); |

Use assembly names

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | var config = new MapperConfiguration(cfg => {      cfg.AddProfiles(new [] {          "Foo.UI",          "Foo.Core"      });  }); |

User marker types for assemblies

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | var config = new MapperConfiguration(cfg => {      cfg.AddProfiles(new [] {          typeof(HomeController),          typeof(Entity)      });  }); |

# Automapper configuration validation

We can validate our mappings using mapper.ConfigurationProvider.AssertConfigurationIsValid() method in unit test.

# Working examples

You can find examples in CashFlow repository in Examples folder.  
<http://tfs.schroders.com:8080/tfs/DefaultCollection/Schroders/Lux%20-%20Global%20TA%20Cashflow%20Notifications/_git/Product.SIMLUX.CashFlowNotification>

# Reference

<http://automapper.readthedocs.io/en/latest/index.html>

# [Writing Unit Test](https://confluence.schroders.com/display/GCFN/Writing+Unit+Test)

* [Overview](https://confluence.schroders.com/display/GCFN/Writing+Unit+Test#WritingUnitTest-Overview)
* [Location of a Unit Test Class](https://confluence.schroders.com/display/GCFN/Writing+Unit+Test#WritingUnitTest-LocationofaUnitTestClass)
* [Naming convention for Unit Tests](https://confluence.schroders.com/display/GCFN/Writing+Unit+Test#WritingUnitTest-NamingconventionforUnitTests)
* [Structure of a Unit Test Class](https://confluence.schroders.com/display/GCFN/Writing+Unit+Test#WritingUnitTest-StructureofaUnitTestClass)
* [Structure of an Unit Test](https://confluence.schroders.com/display/GCFN/Writing+Unit+Test#WritingUnitTest-StructureofanUnitTest)
* [How to Arrange](https://confluence.schroders.com/display/GCFN/Writing+Unit+Test#WritingUnitTest-HowtoArrange)
  + [Scenarios](https://confluence.schroders.com/display/GCFN/Writing+Unit+Test#WritingUnitTest-Scenarios)
  + [Entity Builder](https://confluence.schroders.com/display/GCFN/Writing+Unit+Test#WritingUnitTest-EntityBuilder)
  + [Service Builder](https://confluence.schroders.com/display/GCFN/Writing+Unit+Test#WritingUnitTest-ServiceBuilder)
  + [Service Class Builder](https://confluence.schroders.com/display/GCFN/Writing+Unit+Test#WritingUnitTest-ServiceClassBuilder)
* [How to Act](https://confluence.schroders.com/display/GCFN/Writing+Unit+Test#WritingUnitTest-HowtoAct)
* [How to Assert](https://confluence.schroders.com/display/GCFN/Writing+Unit+Test#WritingUnitTest-HowtoAssert)
* [Testing Mappings](https://confluence.schroders.com/display/GCFN/Writing+Unit+Test#WritingUnitTest-TestingMappings)
  + [Naming Convention](https://confluence.schroders.com/display/GCFN/Writing+Unit+Test#WritingUnitTest-NamingConvention)
  + [Testing Automaps](https://confluence.schroders.com/display/GCFN/Writing+Unit+Test#WritingUnitTest-TestingAutomaps)
  + [Testing Mappings by Property](https://confluence.schroders.com/display/GCFN/Writing+Unit+Test#WritingUnitTest-TestingMappingsbyProperty)

## Overview

Unit testing in this project is taken seriously; this means that unit test code is first class code and underlies the same design principles and rules as other application code. Therefore, a supplementary framework has been designed and needs to be used. The framework and patterns provide:

* we do BDD and not TDD; thus we are not mocking everything away from services, instead we are only mocking boundaries of the process (data loads and writes)
* a generic **test data store** which is used **for all data loads** and writes through implementing a mock repository. Note: the data store is reset after every test run.
* **service calls** of whatever kind **are proxied** by mock implementations using the **test data store**
* the service container, Unity, is setup and configured **only** by test components and test bootstrappers
* domain and data objects are only built by builder objects (i.e. ModelBuilder)
* configuration is read from dummy configuration providers which allow to set values as well (they are reset after each test run)
* FluentAssertions can be used, however do not fall into the Epic Fail of ...Should().Equals()

## Location of a Unit Test Class

A unit test class should be following folder structure of the tested class (entry level). i.e.

if the tested class is → the testing class should be in service/readers/somereadertests

| **Tested Class** | **Testing Class** |
| --- | --- |
| Services/Readers/SomeReader | Service/Readers/SomeReaderTests.cs |

This is suitable for most cases. When there are however too many tests, they can be split by scenario. The file name then should be infixed with the scenario (or abbreviation):

| **Tested Class** | **Testing Class** |
| --- | --- |
| Services/Readers/SomeReader | Service/Readers/SomeReaderTests.LoadTests.cs |

## Naming convention for Unit Tests

<class>\_<method>\_<scenario>  
<class>\_<property>\_<scenario>

Examples:

DateTimeExtensions\_ToDay\_NonUtc  
Car\_Drive\_Ok  
Car\_Drive\_WithoutPilot

<class>\_<property>\_<scenario>

Car\_Pilot  
Car\_Pilot\_ToNull  
Car\_Pilot\_PropertyChange

## Structure of a Unit Test Class

* Can use the TestDependency attribute to ensure that a certain bootstrapper is run before the tests are run  bootstrappers are run as in real-runtime only once.
* The test bootstrapper should make use of the services-bootstrapper and inject mocks for repositories
* There is typically not more than one test bootstrapper per test project.

**Test Class**

|  |
| --- |
| [TestClass]  [TestDependency(typeof(AccountServiceBootstrapper))]  public class AccountServiceTests : TestBase  {          public AccountServiceCatalog Catalog => Container.Resolve<LoggingCatalog>();            [TestMethod]          public async Task LogProviderHubTest\_Log\_LogsIfThresholdNotExceeded()          {            ....          }  } |

**Test Bootstrapper**

|  |
| --- |
| internal class LoggingTestBootstrapper : TestBootstrapper<LoggingBootstrapper>      {          protected override IUnityContainer ConfigureContainer(IUnityContainer container)          {              container = base.ConfigureContainer(container);                container.RegisterType<ILoggingConfiguration, LoggingConfigurationDummy>(new HierarchicalLifetimeManager());                //have to register other providers just to override registration from LoggingBootstrapper and replace it with Mocks              container.RegisterInstance("SqlProvider", new Mock<ILogProvider>().Object);              container.RegisterInstance("ConsoleProvider", new Mock<ILogProvider>().Object);                return container;          }      } |

## Structure of an Unit Test

Unit Tests should be written in Arrange, Act, Assert (AAA) pattern:

* The **Arrange** section of a unit test method initializes objects and sets the value of the data that is passed to the method under test.  
  → The small differences in scenarios should be indicated with comments: // ← has no share class
* The **Act** section invokes the method under test with the arranged parameters.
* The **Assert** section verifies that the action of the method under test behaves as expected. And **only**those. No copy/ paste of asserts is allowed (more is not better!)
* Length should be limited to 15 lines of code

Examples:

**Triple A**

|  |
| --- |
| [TestMethod]  public void Withdraw\_AmountMoreThanBalance\_Throws()  {      // arrange      var account = new Account("John Doe", 10.0);      // act      account.Withdraw(5.0);      // assert      Assert.AreEqual(5, account.Balance);  } |

For more information, please refer to:

<https://msdn.microsoft.com/en-us/library/hh694602.aspx#BKMK_Writing_your_tests>

## How to Arrange

* The arrange part of a unit test should be never more than a couple of lines
* For reasonably complex arrangements (>2 lines), Entity Builders should be used (and written if necessary)
* If an Entity Builder exists for a class, it must be used in all unit tests
* Only absolutely necessary (for the test) values should be arranged in the test. Other defaults need to be covered by the Entity Builder.

**Arrange Act Assert**

|  |  |  |  |
| --- | --- | --- | --- |
| [TestMethod]  public void Withdraw\_AmountMoreThanBalance\_Withdraw\_Ok()  {      // arrange      var account = new AccountBuilder.For("John Doe").WithBalance(10.0).WithMaxDebt(100).New();        // act      account.Withdraw(20.0);       // assert      Assert.AreEqual(-10.0, account.Balance);  } | | | |
| **GOOD** | **BAD** |
| Arrange using entity builders | Create classes using a constructor,   Use a factory method within the unit test class to construct   Use a static factory within the test project |
| Use Moq mocks where the call of a method is to be tested | Write a custom mock with bool indicating the method call |
| Mock only process boundary; not all sibling classes | Mock everything away |
| Follow BDD, test behavior, not implementation specifies | Mocking in depth towards implementation details |
| Enriched with messages for functional asserts | The message assert is not self-explaining |

### Scenarios

The key points of scenarios need to be flagged with a clearly visible and compact comment in order to guide the reader directly to the honey pot.

**Scenarios and comments**

|  |
| --- |
| [TestMethod]  public void Withdraw\_AmountMoreThanBalance\_Withdraw\_DebtOk()  {      // arrange      var account = new AccountBuilder.("John Doe")                           .WithMaxDebt(100).New();  // <-- Debt is high enough          // act      account.Withdraw(20.0);         // assert      Assert.AreEqual(-10.0, account.Balance);  }      [TestMethod]  [ExpectedException(typeof(InvalidOperationException))]  public void Withdraw\_AmountMoreThanBalance\_Withdraw\_DebtExceed()  {      // arrange      var account = new AccountBuilder.("John Doe").WithBalance(10.0)                           .WithMaxDebt(8).New();  // <-- Debt is not high enough      // act      account.Withdraw(20.0);  }    [TestMethod]  [ExpectedException(typeof(InvalidOperationException))]  public void Withdraw\_AmountMoreThanBalance\_Withdraw\_DebtExceed()  {      // arrange      var account = new AccountBuilder.("John Doe").WithBalance(10.0).New();  // <-- Debt is not allowed at all.        // act      account.Withdraw(20.0);  } |

### Entity Builder

An entity builder is a classh which builds a business object, also known as entity. As such it not only constructs the root objects, but all the supplementary items which are normally also available.

| **An Entity Builder is typically** | **An Entity Builder is not** |
| --- | --- |
| Primarily responsible for building one Business Object and its satellites   |  | | --- | | Example: CashFlow entity  + CashFlowStatus object  Example: AladdinBatch + Flow   The actual fabrication can be relayed to another builder, if the relation is another business object | | Building only one data object (it builds the entity and its satellite objects if applicable) |
| Can build supplementary objects and/or use other Entity Builders in order to get those done | a super factory which builds the universe   building of services |
| Does all the necessary orchestration, configuration and registration and other compontents **which are required to keep the application's state consistent for the scenario** | does not reimplement bootstrappers |
| Configures mocks and other services, if necessary |  |
| It has typically two phases:   * 1. The specification submitted to the builder (Fluent syntax)   2. The instantiation of the Business Object: typically by calling the New() or Build() method. |  |
| Implements knowledgable methods to build the object in a simple way | Just a property setter   |  | | --- | | Bad Example:  var flow1 = new AladdinExportBuilder()                  .WithIncidentId("incidentId")                  .InBatch(batch1.Id)                  .InOriginalBatch(batch1b.Id) | |
| **Uses ModelBuilder base class** | Custom entity management |

As such, the Entity Builder replaces:

* Factory methods inside a unit tests
* Static factory methods
* Test Setup sections which are not generic
* Clumsy, long, repeating arrangements by unit tests

### Service Builder

| **An Service Builder is typically** | **An Service Builder is not** |
| --- | --- |
| Constructs complex services which cannot be resolved through Unity to formulate a working test | Used to fabricate services all the way   Used to fabricte the TESTED service. (see Service Class Builder chapter) |
| Only used, if the service configuration setup requires detailed knowledge (on the constructors). Good examples are:   * + Testing of exception handling. |  |
| Used to setup a moqs for corner cases if the ASSERT of the test requries this | Systematically used to construct moqs (this eliminates the benefit of Moq) |

**Entity Builder**

|  |
| --- |
| [TestMethod]  internal sealed AccountBuilder : EntityBuilder<Account>  {      private Identity owner;      private string ownerName;      private decimal balance;      private decimal maxDebt;      public AccountBuilder For(string identityName)      {          this.owner = ServiceCatalog.IdentityRepository.TryGetByName(identityName); // <-- TryGet: returns null, if identity is unknown, see 'AssignAccountIdentity'          this.ownerName = identityName;            return this;      }        public AccountBuilder WithMaxDebt(decimal balance)      {          this.balance = balance;            return this;      }        public AccountBuilder WithBalance(decimal maxDebt)      {          this.maxDebt= maxDebt;            return this;      }        public Account New ()      {          var account = new Account()          account.Balance = balance;          account.MaxDebt = maxDebt;          account.Transactions = new List<Transaction>(); // <-- simple supplementary object, no real use of a dedicated builder here            var accountRepository = ServiceCatalog.AccountRepository.New();          AssignAccountIdentity(account);      }        private void AssignAccountIdentity (Account identity)      {          if (owner != null) { account.Identity = owner; return;}            var identity = new IdentityBuilder().WithName(ownerName).New();          account.Identity = identity;      }  } |

### Service Class Builder

| **GOOD** | **BAD** |
| --- | --- |
| Use Unity to construct a service | Using the constructor directly, with or without parameter lists |
| Write a unit test testing the constructor directly |  |
| Only asserts in scope of the test are written | Asserts of other unit tests are copied to the next one without being changed |
| No need to run the test in order to understand what is asserted | The asserted value is a result of a function and/or not set within the test |
| Enriched with messages for functional asserts | The message assert is not self-explaining |

**Service Class Buildling**

|  |
| --- |
| [TestClass]  [TestDependency(typeof(AccountServiceBootstrapper))]  public class AccountServiceTests : TestBase  {          public AccountServiceCatalog Catalog => Container.Resolve<LoggingCatalog>();            [TestMethod]          public async Task LogProviderHubTest\_Log\_LogsIfThresholdNotExceeded()          {             // Arrange             var hub = Container.Resolve<LogProviderHub>();              ....          }  } |

## How to Act

| **GOOD** | **BAD** |
| --- | --- |
| Is preferably a single line | A sequence of method calls and asserts |

## How to Assert

| **GOOD** | **BAD** |
| --- | --- |
| The assert creates the value for the unit test and should be choosed carefully | No or misleading asserts (note: expected exception are considered as asserts) |
| Only asserts in scope of the test are written | Asserts of other unit tests are copied to the next one without being changed |
| No need to run the test in order to understand what is asserted | The actual value is embedded into the assert as a function |
| Enriched with messages for functional asserts | The message assert is not self-explaining |

## Testing Mappings

All object to object mappings should be performed through AutoMapper (except if there is a very good reason not to do so). Mappings for AutoMapper are to be stored in Profiles, see [Using AutoMapper](https://confluence.schroders.com/display/GCFN/Using+AutoMapper).

### Naming Convention

In order to respect the SOC principle, the naming convention for mapping tests **can** slightly deviate from the above one.

If a test class is testing the entire profile the naming convention remains standard

| **Case** | **Tested Class** | **Testing Class** | **Test Method Prefix** |
| --- | --- | --- | --- |
| Test class is testing the entire profile (this is the standard) | Services/Models/Mapping/FundProfile | Service/Models/Mapping/**FundProfile**Tests.cs | FundProfile\_ |
| Test class contains multiple mappings for not-small objects we can... | Services/Models/Mapping/FundProfile | Service/Models/Mapping/**FundMapping**Tests.cs | Profile\_Fund\_From\_zzz |

The deviations in test method naming convention originate in two points:

1. The tested method is always 'Map' (and is not implemented)
2. The tested method always impacts a source and a target type

### Testing Automaps

In scope of testing, an **auto** map can be simply tested by verifying the configuration

|  |
| --- |
| /// <summary>  /// Verifies that the mapping is syntactically correct.  /// </summary>          [TestMethod]          public void AladdinBatchProfile\_Mapping()          {              var config = new MapperConfiguration(cfg => cfg.AddProfile(new AladdinBatchProfile()));              var mapper = config.CreateMapper();                mapper.ConfigurationProvider.AssertConfigurationIsValid();          } |

### Testing Mappings by Property

For small objects, it is fine to write a single 'Property' test under the following constrains:

* There are less then 8 properties (quite arbitrary arguable number)
* The properties are just copied by the profile
* No logic or transformation applies to any the properties
* In order to validate the mapping, a single test case is sufficient

In **all other cases** **and in cases of doubt**, for each property a test needs to be written.

| **GOOD** | **BAD** |
| --- | --- |
| Create the profile which is tested directly in the test | Use the service catalog's mapper to map (we do not know which implementation we are going to test |
| For small objects, use a property unit test | Using chunky property tests for several scenarios |
| Write mapping tests for each property |  |
| Use builders to construct the source object |  |
| Write tests based using custom values on source object | Write tests based on **Default()** objects |

**Property Test for little classes and simple mappings**

|  |
| --- |
| [TestClass]  public class PricingGroupProfileTests : TestBase  {      /// <summary>      /// Verifies that the values of the properties are correctly mapped.      /// </summary>      [TestMethod]      public void PricingGroupProfile\_Properties()      {          var config = new MapperConfiguration(cfg => cfg.AddProfile(new PricingGroupProfile()));          var mapper = config.CreateMapper();          var expected = new PricingGroupBuilder().WithCode("A").WithOwner("LUA").Build();            var given = mapper.Map<Archive.PricingGroup>(expected);            Assert.AreEqual(expected.Code, given.Code);          Assert.AreEqual(expected.Owner, given.Owner);      }  } |

**Property Mapping Test Case**

|  |
| --- |
| [TestClass]  public class NewTranProfileTests : TestBase  {      private IMapper Mapper => CreateMapper<NewTranProfile>();        [TestMethod]      public void Profile\_LoadNewTran\_From\_Transaction\_AccountId()      {          var flow = new TransactionBuilder().WithAccountId("ALLS10003528").Build();            var given = Mapper.Map<LoadNewTran>(flow);            given.Ntr\_Acct.Should().Equals(flow.AccountId);      }  } |

# [Technical Documentation](https://confluence.schroders.com/display/GCFN/Technical+Documentation)

This article describes the services which are offered by the Cash Flow project and can be helpful during analysis and incident management.

* [Configurations](https://confluence.schroders.com/display/GCFN/Configurations)
* [Design: Aladdin Export](https://confluence.schroders.com/display/GCFN/Design%3A+Aladdin+Export)
* [Design: Posting Portfolio Id Service](https://confluence.schroders.com/display/GCFN/Design%3A+Posting+Portfolio+Id+Service)
* [REST Services](https://confluence.schroders.com/display/GCFN/REST+Services)

# [Design: Aladdin Export](https://confluence.schroders.com/display/GCFN/Design%3A+Aladdin+Export)

* [Entity Services](https://confluence.schroders.com/display/GCFN/Design%3A+Aladdin+Export#Design:AladdinExport-EntityServices)
* [Business Services](https://confluence.schroders.com/display/GCFN/Design%3A+Aladdin+Export#Design:AladdinExport-BusinessServices)
  + [AladdinBatchService:](https://confluence.schroders.com/display/GCFN/Design%3A+Aladdin+Export#Design:AladdinExport-AladdinBatchService:)
  + [AladdinBatchScheduler:](https://confluence.schroders.com/display/GCFN/Design%3A+Aladdin+Export#Design:AladdinExport-AladdinBatchScheduler:)
  + [AladdinFileWriter:](https://confluence.schroders.com/display/GCFN/Design%3A+Aladdin+Export#Design:AladdinExport-AladdinFileWriter:)
  + [AladdinPublisher:](https://confluence.schroders.com/display/GCFN/Design%3A+Aladdin+Export#Design:AladdinExport-AladdinPublisher:)
  + [AladdinExporter:](https://confluence.schroders.com/display/GCFN/Design%3A+Aladdin+Export#Design:AladdinExport-AladdinExporter:)
  + [PortfolioIdResolver:](https://confluence.schroders.com/display/GCFN/Design%3A+Aladdin+Export#Design:AladdinExport-PortfolioIdResolver:)
* [Service Objects](https://confluence.schroders.com/display/GCFN/Design%3A+Aladdin+Export#Design:AladdinExport-ServiceObjects)
  + [Batch:](https://confluence.schroders.com/display/GCFN/Design%3A+Aladdin+Export#Design:AladdinExport-Batch:)
  + [Flow:](https://confluence.schroders.com/display/GCFN/Design%3A+Aladdin+Export#Design:AladdinExport-Flow:)
  + [CashFlow:](https://confluence.schroders.com/display/GCFN/Design%3A+Aladdin+Export#Design:AladdinExport-CashFlow:)
  + [CancelCashFlow:](https://confluence.schroders.com/display/GCFN/Design%3A+Aladdin+Export#Design:AladdinExport-CancelCashFlow:)
* [Endpoint Services](https://confluence.schroders.com/display/GCFN/Design%3A+Aladdin+Export#Design:AladdinExport-EndpointServices)
  + [AladdinExportService:](https://confluence.schroders.com/display/GCFN/Design%3A+Aladdin+Export#Design:AladdinExport-AladdinExportService:)
  + [AladdinRepairService:](https://confluence.schroders.com/display/GCFN/Design%3A+Aladdin+Export#Design:AladdinExport-AladdinRepairService:)

## Entity Services

**AladdinBatchRepository:**

* Allows to create a new AladdinBatch entity, sets created and modified on
* Allows to query AladdinBatches (having AladdinExport items) through the Items queryable
* Allows to save an AladdinBatch
* Allows to read a single AladdinBatch by its Id.

 please note: the repository is the only component, which we will need to mock in our unit tests

## Business Services

### AladdinBatchService:

* Sets the status of the batch on the underlying entity, also:
  + Updates the field 'ModifiedOn' to current date time offset
  + Increments ErrorCount +1 if the status is faulted

### ****AladdinBatchScheduler:****

* Creates the Batches in the database
* Returns the next batch to be exported
* Updates the Batch status:  
  **Queued** when the batch is created which is not having any dummy portfolio ids

### AladdinFileWriter:

* Writes a Batch (service object!) into a cash flow or a cancel file in a temporary directory (on the machine)
* Updates the Batch status:  
  **Exporting**when writing is started → **Exported**when writing is successfully completed (please note that the file name is part of the Batch)  
  **Exporting**when writing is started → **Faulted**when writing was failing for some reason (exception needs to contain the batch id, filename and potentially inner exception content)

### AladdinPublisher:

* Copies the file from the temporary directory to the Aladdin file share (and later to an archive folder)
* Updates the Batch status:  
  **Publishing**when writing is started → **Published**when copy is successfully completed (please note that the file name is part of the Batch)  
  **Publishing**when writing is started → **Faulted**when writing was failing for some reason (exception needs to contain the batch id, filename and potentially inner exception content)

### AladdinExporter:

Service which orchestrates the other services, called by the endpoint(s)

1. Lets the Batch Schedule schedule
2. In a loop, tries to get the next batch from the scheduler, if positive:
   1. Lets the file write write the file
   2. Lets the AladdinPublisher publish the file to the file share

### PortfolioIdResolver:

* Reads all flows without mandates or without posting portfolio ids and attempts resolving them (static data might have been changed since last call)

## Service Objects

### Batch:

Represents a sequence of cash flows or cancellations which can be written in a single file.

### Flow:

Represents an abstract flow which relates for an Aladdin Cash Posting Portfolio

### CashFlow:

Represents a cash-in or cash-out movement.

### CancelCashFlow:

Represents the cancellation of a cash movement.

## Endpoint Services

### AladdinExportService:

* REST endpoint connector for manual exports and Quartz service host

### AladdinRepairService:

* REST endpoint connector for manual repairs and Quartz service host

# [Design: Posting Portfolio Id Service](https://confluence.schroders.com/display/GCFN/Design%3A+Posting+Portfolio+Id+Service)

## Retrieving the Portfolio Ids

The portfolio id mappings are provided by a service from the London team.

[*http://bdm\_devdit:3030/api/LuxPostingPortfolios*](http://bdm_devdit:3030/api/LuxPostingPortfolios)

**Portfolio Service Response object** Expand source

### Details:

* The service will present the full set of IRIS data each time it is called
* While it can be called as frequently as desired, the underlying data will only be updated once a day
* It was agreed that if for any reason the service is unavailable this must **not**impact upon the FundSphere cash flows extract to Aladdin – i.e. Fundsphere should continue to use the most recent version of the posting portfolio data it has until the service is back online and an update is available. (It is recognised this will have an impact on any new mandates or changes to existing – exception flows are to be defined)
* The service uses Windows Authentication, it is read-only and is not expected to require any additional security. There are not believed to be any confidentiality issues in sharing the required information with the FundSphere system
* **IRIS\_LegalMandate\_Name** and **IRIS\_Pfolio\_TradingName** are not used by FundSphere. It is recommended these are dropped from the service to help further anonymise the data - unless Lux business teams have any objections?
* The values for IRIS\_Pfolio\_MandatePostingPortfolio are:

| **Values** | **Action** |
| --- | --- |
| CA - Red | ignore |
| CA - Sub | ignore |
| CA - Sub/Red | ignore |
| CN - Red | ignore |
| CN - Sub | ignore |
| **CN - Sub/Red** | use |

* For FundSphere, only the higlighted lines are relevant (the service provider will infact remove the other lines)

# [REST Services](https://confluence.schroders.com/display/GCFN/REST+Services)

## [REST: Aladdin Export](https://confluence.schroders.com/display/GCFN/REST%3A+Aladdin+Export)

## [REST: Log Integration](https://confluence.schroders.com/display/GCFN/REST%3A+Log+Integration)

## [REST: Load Cash Flows](https://confluence.schroders.com/display/GCFN/REST%3A+Load+Cash+Flows)

## [REST: Load Collateral Movement](https://confluence.schroders.com/display/GCFN/REST%3A+Load+Collateral+Movement)

## [REST: Maintenance](https://confluence.schroders.com/display/GCFN/REST%3A+Maintenance)

## [REST: Static Data](https://confluence.schroders.com/display/GCFN/REST%3A+Static+Data)

## [REST: Sanity Tests](https://confluence.schroders.com/display/GCFN/REST%3A+Sanity+Tests)

See also:

[Fundsphere and POST - Restlets](https://confluence.schroders.com/display/LAP/Fundsphere+and+POST+-+Restlets)

# [REST: Aladdin Export](https://confluence.schroders.com/display/GCFN/REST%3A+Aladdin+Export)

* Created by [Pusz, Szymon](https://confluence.schroders.com/display/~puszszy" \o "), last modified on [Jan 31, 2019](https://confluence.schroders.com/pages/diffpagesbyversion.action?pageId=36108047&selectedPageVersions=11&selectedPageVersions=12)

[Go to start of metadata](https://confluence.schroders.com/display/GCFN/REST%3A+Aladdin+Export#page-metadata-start)

* [Check whether data is available to be exported (Get export queue)](https://confluence.schroders.com/display/GCFN/REST%3A+Aladdin+Export#REST:AladdinExport-Checkwhetherdataisavailabletobeexported(Getexportqueue))
* [Repair unresolved cash posting ids](https://confluence.schroders.com/display/GCFN/REST%3A+Aladdin+Export#REST:AladdinExport-RepairunresolvedcashpostingidsrepairAladdinCashId)
* [Manually export flows to Aladdin](https://confluence.schroders.com/display/GCFN/REST%3A+Aladdin+Export#REST:AladdinExport-ManuallyexportflowstoAladdinexportAladdinFlows)
* [Manually export flows of incidents to Aladdin](https://confluence.schroders.com/display/GCFN/REST%3A+Aladdin+Export#REST:AladdinExport-ManuallyexportflowsofincidentstoAladdinexportAladdinFlows)
* [Archive Aladdin export flows](https://confluence.schroders.com/display/GCFN/REST%3A+Aladdin+Export#REST:AladdinExport-ArchiveAladdinexportflowsarchiveAladdinExport)

### Check whether data is available to be exported (Get export queue)

| **GET** | **api/feeds/Aladdin/stats** |
| --- | --- |
|  |  |

Example Requests:

DEV - http://naos-dev.schroders.com:19081/SIMProduct.FundSphere/FundSphere.Orchestrator/api/feeds/Aladdin

UAT - http://naos-uat.schroders.com:19081/SIMProduct.FundSphere/FundSphere.Orchestrator/api/feeds/Aladdin

**Example Result** Expand source

### Repair unresolved cash posting ids

| **GET** | **api/feeds/Aladdin/repair?incidentId=123abc&tradeDate={date}&fundSphereBatchNumber={string}&aladdinExportId={number}** |
| --- | --- |
| incidentId | Incident number (mandatory) |
| tradeDate | Trade date (optional) |
| fundSphereBatchNumber | FundSphere batch number (optional) |
| aladdinExportId | AladdinExportId from table flows.AladdinExport (optional) |

Examples:

DEV - http://naos-dev.schroders.com:19081/SIMProduct.FundSphere/FundSphere.Orchestrator/api/feeds/Aladdin/repair?incidentId=INC123456&tradeDate=2018-07-19&fundSphereBatchNumber=180719001113300&aladdinExportId=75774

UAT - http://naos-uat.schroders.com:19081/SIMProduct.FundSphere/FundSphere.Orchestrator/api/feeds/Aladdin/repair?incidentId=INC123456&tradeDate=2018-07-19&fundSphereBatchNumber=180719001113300&aladdinExportId=75774

### Manually export flows to Aladdin

 This service is usually scheduled to run every 10 seconds. It should therefore only be called, if the scheduler fails.

| **GET** | **api/feeds/Aladdin/export** |
| --- | --- |
|  |  |

Example Requests:

DEV - http://naos-dev.schroders.com:19081/SIMProduct.FundSphere/FundSphere.Orchestrator/api/feeds/Aladdin/export

UAT - http://naos-uat.schroders.com:19081/SIMProduct.FundSphere/FundSphere.Orchestrator/api/feeds/Aladdin/export

**Example Result** Expand source

### Manually export flows of incidents to Aladdin

 This service is usually scheduled to run every 10 seconds. It should therefore only be called, if the scheduler fails.

| **GET** | **api/feeds/Aladdin/export/incidents** |
| --- | --- |
| isDryRun | Determines whether the flows marked by the incident should be exported to the dry run location, or the real Aladdin upload space |

See also [Re-export a flow which was skipped for bad reason to Aladdin](https://confluence.schroders.com/display/GCFN/Re-export+a+flow+which+was+skipped+for+bad+reason+to+Aladdin)

Example Requests:

DEV - http://naos-dev.schroders.com:19081/SIMProduct.FundSphere/FundSphere.Orchestrator/api/feeds/Aladdin/export/incidents/IN232456

UAT - http://naos-uat.schroders.com:19081/SIMProduct.FundSphere/FundSphere.Orchestrator/api/feeds/Aladdin/export/incidents/IN232443

**Example Result** Expand source

### Archive Aladdin export flows

| **GET** | **api/feeds/Aladdin/archive** |
| --- | --- |
|  |  |

Examples:

DEV - http://naos-dev.schroders.com:19081/SIMProduct.FundSphere/FundSphere.Orchestrator/api/feeds/Aladdin/archive

UAT - http://naos-uat.schroders.com:19081/SIMProduct.FundSphere/FundSphere.Orchestrator/api/feeds/Aladdin/archive

# [REST: Log Integration](https://confluence.schroders.com/display/GCFN/REST%3A+Log+Integration)

### Submit a process result

This service is used by FundSphere to submit/ put a FundSphere process result to our internal log systems. The service will forward the logs and update dashboard according to the provided input.

|  |  |
| --- | --- |
| PUT | /SIMProduct.FundSphere/FundSphere.Orchestrator/api/fncf/monitoring |
| Request Body | Instance of ProcessStatus |
| Response Body | === |
| HTTP Codes | 200 or 500 |

**Example Request** Expand source

Examples:

DEV - PUT http://naos-dev.schroders.com:19081/SIMProduct.FundSphere/FundSphere.Orchestrator/api/fncf/monitoring

SIT - PUT http://naos-sit.schroders.com:19081/SIMProduct.FundSphere/FundSphere.Orchestrator/api/fncf/monitoring

### Check endpoint status

This service is used by FundSphere to check whether the process endpoint is online or not. Its status is shown in the System Status dialog.

|  |  |
| --- | --- |
| GET | /SIMProduct.FundSphere/FundSphere.Orchestrator/api/fncf/monitoring |
| Request Body | === |
| Response Body | === |
| HTTP CODES | 200 or 500 |

# [REST: Load Cash Flows](https://confluence.schroders.com/display/GCFN/REST%3A+Load+Cash+Flows)

* [Read the pre cut-off feeds from SIMLUX](https://confluence.schroders.com/display/GCFN/REST%3A+Load+Cash+Flows#REST:LoadCashFlows-Readtheprecut-offfeedsfromSIMLUX)
* [Read the post cut-off feeds from SIMLUX](https://confluence.schroders.com/display/GCFN/REST%3A+Load+Cash+Flows#REST:LoadCashFlows-Readthepostcut-offfeedsfromSIMLUX)
* [Read the confirmed feeds from SIMLUX](https://confluence.schroders.com/display/GCFN/REST%3A+Load+Cash+Flows#REST:LoadCashFlows-ReadtheconfirmedfeedsfromSIMLUX)
* [Read the correction feeds from SIMLUX](https://confluence.schroders.com/display/GCFN/REST%3A+Load+Cash+Flows#REST:LoadCashFlows-ReadthecorrectionfeedsfromSIMLUX)
* [Load the pre cut-off feeds from SIMLUX into FundSphere Staging](https://confluence.schroders.com/display/GCFN/REST%3A+Load+Cash+Flows#REST:LoadCashFlows-Loadtheprecut-offfeedsfromSIMLUXintoFundSphereStaging)
* [Load the post cut-off feeds from SIMLUX into FundSphere Staging](https://confluence.schroders.com/display/GCFN/REST%3A+Load+Cash+Flows#REST:LoadCashFlows-Loadthepostcut-offfeedsfromSIMLUXintoFundSphereStaging)
* [Load the confirmed feeds from SIMLUX into FundSphere Staging](https://confluence.schroders.com/display/GCFN/REST%3A+Load+Cash+Flows#REST:LoadCashFlows-LoadtheconfirmedfeedsfromSIMLUXintoFundSphereStaging)
* [Load the correction feeds from SIMLUX into FundSphere Staging](https://confluence.schroders.com/display/GCFN/REST%3A+Load+Cash+Flows#REST:LoadCashFlows-LoadthecorrectionfeedsfromSIMLUXintoFundSphereStaging)
* [Obsolete: Load SIMLUX Cash Flow into IMDB](https://confluence.schroders.com/display/GCFN/REST%3A+Load+Cash+Flows#REST:LoadCashFlows-Obsolete:LoadSIMLUXCashFlowintoIMDB)

### Read the pre cut-off feeds from SIMLUX

| **GET** | **api/feeds/SIMLUX/precutoff?tradeDate={tradeDate}&priceGroup={priceGroup}&legalFundCode={legalFundCode}** |
| --- | --- |
| pricingGroup | pricing group, ex 'SCHLUX ' |
| legalFundCode | Optional: legal fund code, ex. 'IQGG' |
| tradeDate | bargain date in**yyyy-MM-dd f**ormat (invariant culture) |

 Calling this service will not change/ modify any data. This service is for diagnostic reasons only.

Examples:

DEV - http://naos-dev.schroders.com:19081/SIMProduct.FundSphere/FundSphere.Orchestrator/api/feeds/SIMLUX/precutoff?tradeDate=2016-05-09&priceGroup=SCHLUX

**Example Result** Expand source

### Read the post cut-off feeds from SIMLUX

| **GET** | **api/feeds/SIMLUX/postcutoff?tradeDate={tradeDate}&priceGroup={priceGroup}&legalFundCode={legalFundCode}** |
| --- | --- |
| priceGroup | pricing group, ex 'SCHLUX ' |
| legalFundCode | Optional: legal fund code, ex. 'IQGG' |
| tradeDate | bargain date in**yyyy-MM-dd f**ormat (invariant culture) |

 Calling this service will not change/ modify any data. This service is for diagnostic reasons only.

Examples:

DEV - http://naos-dev.schroders.com:19081/SIMProduct.FundSphere/FundSphere.Orchestrator/api/feeds/SIMLUX/postcutoff?tradeDate=2016-05-09&priceGroup=SCHLUX

**Example Result** Expand source

### Read the confirmed feeds from SIMLUX

| **GET** | **api/feeds/SIMLUX/confirmed?tradeDate={tradeDate}&priceGroup={priceGroup}** |
| --- | --- |
| priceGroup | pricing group, ex 'SCHLUX ' |
| tradeDate | bargain date in**yyyy-MM-dd f**ormat (invariant culture) |

 Calling this service will not change/ modify any data. This service is for diagnostic reasons only.

Examples:

DEV - http://naos-dev.schroders.com:19081/SIMProduct.FundSphere/FundSphere.Orchestrator/api/feeds/SIMLUX/confirmed?tradeDate=2016-05-09&priceGroup=SCHLUX

**Example Result** Expand source

### Read the correction feeds from SIMLUX

| **GET** | **api/feeds/SIMLUX/correction?tradeDate={tradeDate}&priceGroup={priceGroup}** |
| --- | --- |
| priceGroup | pricing group, ex 'SCHLUX ' |
| tradeDate | bargain date in**yyyy-MM-dd f**ormat (invariant culture) |
| applyChangeDetection | optional: true or false. If false, the correction loader will not filter out flows were the amount did not change. |

 Calling this service will not change/ modify any data. This service is for diagnostic reasons only.

Examples:

DEV - http://naos-dev.schroders.com:19081/SIMProduct.FundSphere/FundSphere.Orchestrator/api/feeds/SIMLUX/correction?tradeDate=2016-05-09&priceGroup=SCHLUX

**Example Result** Expand source

### Load the pre cut-off feeds from SIMLUX into FundSphere Staging

| **GET** | **api/feeds/SIMLUX/precutoff?tradeDate={tradeDate}&priceGroup={priceGroup}&legalFundCode={legalFundCode}** |
| --- | --- |
| priceGroup | pricing group, ex 'SCHLUX ' |
| legalFundCode | Optional: legal fund code, ex. 'IQGG' |
| tradeDate | bargain date in**yyyy-MM-dd f**ormat (invariant culture) |

Example Requests:

DEV - http://naos-dev.schroders.com:19081/SIMProduct.FundSphere/FundSphere.Orchestrator/api/feeds/SIMLUX/precutoff?tradeDate=2016-05-09&priceGroup=SCHLUX

**Example Result** Expand source

### Load the post cut-off feeds from SIMLUX into FundSphere Staging

| **GET** | **api/feeds/SIMLUX/postcutoff/load?tradeDate={tradeDate}&priceGroup={priceGroup}&legalFundCode={legalFundCode}** |
| --- | --- |
| priceGroup | pricing group, ex 'SCHLUX ' |
| legalFundCode | Optional: legal fund code, ex. 'IQGG' |
| tradeDate | bargain date in**yyyy-MM-dd f**ormat (invariant culture) |

Example Requests:

DEV - http://naos-dev.schroders.com:19081/SIMProduct.FundSphere/FundSphere.Orchestrator/api/feeds/SIMLUX/postcutoff?tradeDate=2016-05-09&priceGroup=SCHLUX

**Example Result** Expand source

### Load the confirmed feeds from SIMLUX into FundSphere Staging

| **GET** | **api/feeds/SIMLUX/confirmed/load?tradeDate={tradeDate}&priceGroup={priceGroup}** |
| --- | --- |
| priceGroup | pricing group, ex 'SCHLUX ' |
| tradeDate | bargain date in**yyyy-MM-dd f**ormat (invariant culture) |

Example Requests:

DEV - http://naos-dev.schroders.com:19081/SIMProduct.FundSphere/FundSphere.Orchestrator/api/feeds/SIMLUX/confirmed?tradeDate=2016-05-09&priceGroup=SCHLUX

**Example Result** Expand source

### Load the correction feeds from SIMLUX into FundSphere Staging

| **GET** | **api/feeds/SIMLUX/correction/load?tradeDate={tradeDate}&priceGroup={priceGroup}** |
| --- | --- |
| priceGroup | pricing group, ex 'SCHLUX ' |
| tradeDate | bargain date in**yyyy-MM-dd f**ormat (invariant culture) |
| applyChangeDetection | optional: true or false. If false, the correction loader will not filter out flows were the amount did not change. |

Example Requests:

DEV - http://naos-dev.schroders.com:19081/SIMProduct.FundSphere/FundSphere.Orchestrator/api/feeds/SIMLUX/correction?tradeDate=2016-05-09&priceGroup=SCHLUX

**Example Result** Expand source

### Obsolete: Load SIMLUX Cash Flow into IMDB

| **GET** | **api/TriggerLoadTransferAgencyLU/{type}/{pricingGroup}/{bargainDate}** |
| --- | --- |
| type | one of PreCutOff, PostCutOff, Priced or Correction |
| pricingGroup | pricing group, ex 'SCHLUX ' |
| bargainDate | bargain date in**yyyy-MM-dd f**ormat invariant culture) |

 This service is **obsolete.**

 The Metrosoft IMDB → TOTACC job has to be run in order to be able to see the data in FundSphere

Examples:

DEV - http://naos-dev.schroders.com:19081/SIMProduct.FundSphere/FundSphere.Orchestrator/api/TriggerLoadTransferAgencyLU/PreCutOff/SCHLUX/2018-01-15

SIT - http://naos-sit.schroders.com:19081/SIMProduct.FundSphere/FundSphere.Orchestrator/api/TriggerLoadTransferAgencyLU/PreCutOff/SCHLUX/2018-01-15

# [REST: Load Collateral Movement](https://confluence.schroders.com/display/GCFN/REST%3A+Load+Collateral+Movement)

### Manually Load Collateral Movement message into Staging

| **GET** | **api/feeds/collateralMovements/load?tradeDate={tradeDate}** |
| --- | --- |
| TradeDate | Optional parameter, defaulting to 'today'. Format is invariant culture. |

Response:

|  |
| --- |
| {    "LoadedCollateralMovements" : 1234  } |

# [REST: Maintenance](https://confluence.schroders.com/display/GCFN/REST%3A+Maintenance)

* Created by [Potts, Harvey](https://confluence.schroders.com/display/~pottsha), last modified by [Pusz, Szymon](https://confluence.schroders.com/display/~puszszy" \o ") on [Dec 12, 2018](https://confluence.schroders.com/pages/diffpagesbyversion.action?pageId=32490278&selectedPageVersions=8&selectedPageVersions=9)

[Go to start of metadata](https://confluence.schroders.com/display/GCFN/REST%3A+Maintenance#page-metadata-start)

### Get all jobs

| **GET** | **api/jobs** |
| --- | --- |
|  |  |
| RETURNS | A list of job infos. |

**Example result** Expand source

Examples:

DEV - http://naos-dev.schroders.com:19081/SIMProduct.FundSphere/FundSphere.Orchestrator/api/jobs/

SIT - http://naos-sit.schroders.com:19081/SIMProduct.FundSphere/FundSphere.Orchestrator/api/jobs/

### Disable all jobs

| **GET** | **api/jobs/disable** |
| --- | --- |
|  |  |

Examples:

DEV - http://naos-dev.schroders.com:19081/SIMProduct.FundSphere/FundSphere.Orchestrator/api/jobs/disable

SIT - http://naos-sit.schroders.com:19081/SIMProduct.FundSphere/FundSphere.Orchestrator/api/jobs/disable

### Enable all jobs

| **GET** | **api/jobs/enable** |
| --- | --- |
|  |  |

Examples:

DEV - http://naos-dev.schroders.com:19081/SIMProduct.FundSphere/FundSphere.Orchestrator/api/jobs/enable

SIT - http://naos-sit.schroders.com:19081/SIMProduct.FundSphere/FundSphere.Orchestrator/api/jobs/enable

### Disable a job

| **GET** | **api/jobs/{jobkey}/disable** |
| --- | --- |
| jobkey | The key of the job, returned by Get-All-Jobs (or in config) |
| RETURNS | The job info |

Examples:

DEV - http://naos-dev.schroders.com:19081/SIMProduct.FundSphere/FundSphere.Orchestrator/api/jobs/FundSphere.ExportAladdin/disable

SIT - http://naos-sit.schroders.com:19081/SIMProduct.FundSphere/FundSphere.Orchestrator/api/jobs/FundSphere.ExportAladdin/disable

**Example result** Expand source

### Enable a job

| **GET** | **api/jobs/{jobkey}/enable** |
| --- | --- |
| jobkey | The key the job, returned by Get-All-Jobs (or in config) |
| RETURNS | The job info |

Examples:

DEV - http://naos-dev.schroders.com:19081/SIMProduct.FundSphere/FundSphere.Orchestrator/api/jobs/FundSphere.ExportAladdin/enable

SIT - http://naos-sit.schroders.com:19081/SIMProduct.FundSphere/FundSphere.Orchestrator/api/jobs/FundSphere.ExportAladdin/enable

**Example result** Expand source

### Trigger Metrosoft loader

| **GET** | **api/jobs/metrosoft/{jobkey}** |
| --- | --- |
| jobkey | The key the job, returned by Get-All-Jobs (or in config) |
| RETURNS | HTTP 200 - job triggered or already running  HTTP 404 - specified job not found  HTTP 400 - JobKey doesn't starts with "MetroSoft." prefix  HTTP 500 - Unhandled error occured |

Examples:

DEV - http://naos-dev.schroders.com:19081/SIMProduct.FundSphere/FundSphere.Orchestrator/api/jobs/metrosoft/MetroSoft.Reload.IMDB

SIT - http://naos-sit.schroders.com:19081/SIMProduct.FundSphere/FundSphere.Orchestrator/api/jobs/metrosoft/MetroSoft.Reload.IMDB

# [REST: Static Data](https://confluence.schroders.com/display/GCFN/REST%3A+Static+Data)

The loaders below load static data into FundSphere Staging on-demand. Please note that they are normally run by schedule.

* [FSI v7.x and higher](https://confluence.schroders.com/display/GCFN/REST%3A+Static+Data#REST:StaticData-FSIv7.xandhigher)
  + [Manually Load Exchange Rates](https://confluence.schroders.com/display/GCFN/REST%3A+Static+Data#REST:StaticData-ManuallyLoadExchangeRates)
  + [Manually Load Fund Hierarchy](https://confluence.schroders.com/display/GCFN/REST%3A+Static+Data#REST:StaticData-ManuallyLoadFundHierarchy)
  + [Manually Load Prices and NAVs](https://confluence.schroders.com/display/GCFN/REST%3A+Static+Data#REST:StaticData-ManuallyLoadPricesandNAVs)
* [FSI v6.60](https://confluence.schroders.com/display/GCFN/REST%3A+Static+Data#REST:StaticData-FSIv6.60)
  + [Manually Load Exchange Rates](https://confluence.schroders.com/display/GCFN/REST%3A+Static+Data#REST:StaticData-ManuallyLoadExchangeRates.1)
  + [Manually Load Fund Hierarchy](https://confluence.schroders.com/display/GCFN/REST%3A+Static+Data#REST:StaticData-ManuallyLoadFundHierarchy.1)
  + [Manually Load Prices and NAVs](https://confluence.schroders.com/display/GCFN/REST%3A+Static+Data#REST:StaticData-ManuallyLoadPricesandNAVs.1)

## FSI v7.x and higher

### Manually Load Exchange Rates

| **GET** | **api/exchangeRates/load?date=2018-06-14** |
| --- | --- |
| date | Date for which the exchange rates should be loaded. |

### Manually Load Fund Hierarchy

| **GET** | **api/fundHierarchy/load** |
| --- | --- |
|  |  |

### Manually Load Prices and NAVs

| **GET** | **api/navs/load?date=2018-05-30** |
| --- | --- |
| date | Date for which the exchange rates should be loaded. |

## FSI v6.60

 These services are obsolete

### Manually Load Exchange Rates

| **GET** | **api/TriggerLoadExchangeRate/2018-06-14** |
| --- | --- |
|  |  |

### Manually Load Fund Hierarchy

| **GET** | **api/triggerfundsload** |
| --- | --- |
|  |  |

### Manually Load Prices and NAVs

| **GET** | **api/static/pricesnav/load?date=2018-05-30** |
| --- | --- |
| date | Date for which the exchange rates should be loaded. |

# [REST: Sanity Tests](https://confluence.schroders.com/display/GCFN/REST%3A+Sanity+Tests)

* [Run Sanity Tests for Aladdin Export](https://confluence.schroders.com/display/GCFN/REST%3A+Sanity+Tests#REST:SanityTests-RunSanityTestsforAladdinExport)

### Run Sanity Tests for Aladdin Export

| **GET** | **api/feeds/Aladdin/runSanityTest** |
| --- | --- |
| data | Executes prepared scenarios on aladdin export |

Example Requests: (Currently works only on SIT)

|  |  |
| --- | --- |
| GET | http://naos-sit.schroders.com:19081/SIMProduct.FundSphere/FundSphere.Orchestrator/api/feeds/Aladdin/runSanityTest |

# [C# Guidelines and Policies](https://confluence.schroders.com/display/GCFN/C%23+Guidelines+and+Policies)

This article describes general guidelines for the usage of C# inside the CashFlow project:

* [Naming Convention](https://confluence.schroders.com/display/GCFN/C%23+Guidelines+and+Policies#C#GuidelinesandPolicies-NamingConvention)
  + [Project Names](https://confluence.schroders.com/display/GCFN/C%23+Guidelines+and+Policies#C#GuidelinesandPolicies-ProjectNames)
  + [Namespaces](https://confluence.schroders.com/display/GCFN/C%23+Guidelines+and+Policies#C#GuidelinesandPolicies-Namespaces)
  + [Service Fabric Service Naming conventions](https://confluence.schroders.com/display/GCFN/C%23+Guidelines+and+Policies#C#GuidelinesandPolicies-ServiceFabricServiceNamingconventions)
* [Warnings as errors](https://confluence.schroders.com/display/GCFN/C%23+Guidelines+and+Policies#C#GuidelinesandPolicies-Warningsaserrors)
* [Parameter Checks](https://confluence.schroders.com/display/GCFN/C%23+Guidelines+and+Policies#C#GuidelinesandPolicies-ParameterChecks)
* [Code Blocks:](https://confluence.schroders.com/display/GCFN/C%23+Guidelines+and+Policies#C#GuidelinesandPolicies-CodeBlocks:)
* [Comments](https://confluence.schroders.com/display/GCFN/C%23+Guidelines+and+Policies#C#GuidelinesandPolicies-Comments)
* [Formatted Strings](https://confluence.schroders.com/display/GCFN/C%23+Guidelines+and+Policies#C#GuidelinesandPolicies-FormattedStrings)
* [Recommended 3rd Party Libraries](https://confluence.schroders.com/display/GCFN/C%23+Guidelines+and+Policies#C#GuidelinesandPolicies-Recommended3rdPartyLibraries)

Entries which are ~~strikethrough~~ are either not agreed or communicated yet.

## Naming Convention

Basically, the .NET convention applies Private class members may be prefixed (e.g. '\_') or not, but the prefix-strategy must be applied on **all**projects or none. Additionally, the following naming rules apply:

* Length of the variable/ method/ service: as short as possible and as long as necessary (in order to be specific).
* Names should not describe the implementation, but the high level function
* For common operations, common verbs have to be used (when a record is created, use 'Create' verb). If multiple versions exist, one consistent version is used (New() or Create() can be synonyms, but synonyms should be avoided)

### Project Names

Namespaces within the CashFlow project need to be harmonized and follow .NET standards (company/project/component)

SIMLUX.Product.CashFlow.LoadFund

### Namespaces

Namespaces within the CashFlow project need to be harmonized and follow .NET standards (company/project/component). As per interpretation with capability lead, namespaces within the TA program should be starting with:

SIMLUX.Product  
  
Example for fund hierarchy loader:  
SIMLUX.Product.CashFlow

Namespaces should embrace items of the complete solution and not by C# project; this is why the component discriminator often does not contain the component.

### Service Fabric Service Naming conventions

Service Naming conventions are the following

* Method names should have **distinct names** (method override is not supported)

Service Fabric contracts are RDP calls and therefore underlie certain limitations.

* Method overloads are not supported as methods with same name cannot be distinguished from each other (in contrast to C# compiler and linker)
* Namespace changes are only supported within applications, if the contract is used outside of the application, the routing would be broken.

## ~~Warnings as errors~~

All projects must elevate the following warnings to errors:

* Type your task here, using "@" to assign to a user and "//" to select a due date

## Parameter Checks

All parameters of non-private are checked prior usage. The check can happen through the Guard, but not exclusively: more complex checks may be added after the simple guards. With guards, the following typical checks should be done

* Value is not null, not null + not empty, not null  + not empty + not white-space
* Value is with range: bigger than, smaller than, between

(of course the implementation of the checks above differs with the data type -and not all checks are available for all types)

## Code Blocks:

Usage of code blocks is mandatory for all tokens which don't enforce it which are at least:

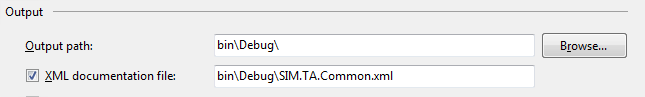
* if...else...else if
* while...do
* do...while
* foreach...
* for ...

Rationale: not using code block saves a second, but an accidental merge can drop the line -or put something in between the single line which can change the behavior of the logic completely. Using code blocks not only eliminates this problem, but increases readability (can be collapsed, is highlighted by editor) and extensibility (new lines can be directly added).

| **DO** | **DON'T** |
| --- | --- |
| https://confluence.schroders.com/download/attachments/29769559/image2017-12-29_14-25-29.png?version=1&modificationDate=1514556591290&api=v2 | https://confluence.schroders.com/download/attachments/29769559/image2017-12-29_14-25-6.png?version=1&modificationDate=1514556591323&api=v2 |

## Comments

It is important that all public/ internal (non-private) members have meaningful comments. Meaningful translates to:

* A summary is present
* Non-trivial parameters are explained with a <param> node
* Directly thrown exceptions are listed, including the cases when this happens -this explicitly also applies to contracts
* Command the **why** and not the **what**(see common development guidelines)
* A documentation file must be created for each assembly  
  

Be aware about the Integration Operation Segregation Principle (IOSP). The need for extensive comments indicates that the logic should be moved into a dedicated method.

| **DO** | **DON'T** |
| --- | --- |
| https://confluence.schroders.com/download/attachments/29769559/image2017-12-29_14-24-24.png?version=1&modificationDate=1514556591337&api=v2 | https://confluence.schroders.com/download/attachments/29769559/image2017-12-29_14-22-6.png?version=1&modificationDate=1514556591383&api=v2  (empty comment) |
|  |  |

## Formatted Strings

With C# 7.0 usage of formatted strings is easier and became more readable and made String.Format mostly avoidable:

|  |  |
| --- | --- |
| **DO** | https://confluence.schroders.com/download/attachments/29769559/image2017-12-29_14-32-9.png?version=1&modificationDate=1514556591277&api=v2 |
| **DON'T** | https://confluence.schroders.com/download/attachments/29769559/image2017-12-29_14-32-42.png?version=1&modificationDate=1514556591260&api=v2 |
| **DON'T** | https://confluence.schroders.com/download/attachments/29769559/image2017-12-29_14-33-4.png?version=1&modificationDate=1514556591227&api=v2 |

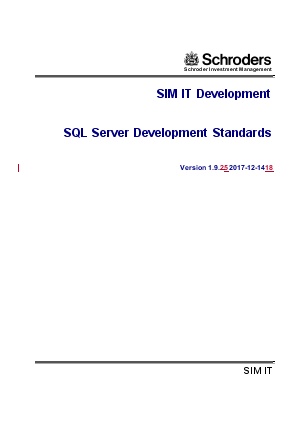
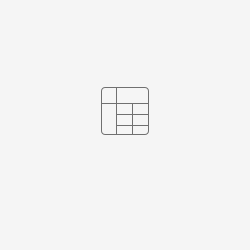
## Recommended 3rd Party Libraries

In order to keep development homogenous, duplicate and overlapping features of third party libraries should be avoided as much possible. In this mission, the following libraries should be preferred over others with similar feature set:

| **Name** | **Assembly/ Package Name** | **Description** |
| --- | --- | --- |
| **Unity** | Microsoft.Practices.Unity.dll | Inversion of control container |
| **AutoMapper** |  | Type adapter |
| **Moq** |  | Mocking framework, for unit testing only |
| **Json** | Newtonsoft.Json.dll | Json serializer/ deserializer |
| **Quartz** | Quartz.dll | Job Scheduler (typically inside a windows process) |
| **Serilog** | Serilog.dll | Logging engine |

# [SQL Policies](https://confluence.schroders.com/display/GCFN/SQL+Policies)

For the FundSphere project, the standard SQL policies and guidelines apply. The QA calendar can also be found here.

[](https://confluence.schroders.com/download/attachments/29764133/SQL%20Server%20Development%20Standards.doc?version=1&modificationDate=1513240763717&api=v2) Document[[](https://confluence.schroders.com/download/attachments/29764133/Development%20Team%20QA%20Calendar.xlsx?version=2&modificationDate=1513268905457&api=v2)Development Team QA Calendar.xlsx](https://confluence.schroders.com/download/attachments/29764133/Development%20Team%20QA%20Calendar.xlsx?version=2&modificationDate=1513268905457&api=v2)

Then however, in our commitment to service oriented design we need to limit artificially the responsibilities which may rely in stored procedures:

* [DO NOT: Include business logic in stored procedures](https://confluence.schroders.com/display/GCFN/SQL+Policies#SQLPolicies-DONOT:Includebusinesslogicinstoredprocedures)
* [DO NOT: use strings as command text](https://confluence.schroders.com/display/GCFN/SQL+Policies#SQLPolicies-DONOT:usestringsascommandtext)
* [AVOID: Stored Procedures in not owned databases](https://confluence.schroders.com/display/GCFN/SQL+Policies#SQLPolicies-AVOID:StoredProceduresinnotowneddatabases)
* [DO: Use string resource as command text](https://confluence.schroders.com/display/GCFN/SQL+Policies#SQLPolicies-SqlResourceTextDO:Usestringresourceascommandtext)
* [DO: Include stored procedures into DACPACs](https://confluence.schroders.com/display/GCFN/SQL+Policies#SQLPolicies-DO:IncludestoredproceduresintoDACPACs)

For deployment automation purposes, usage of stored procedures have to be reviewed completely - especially when the database is not under this teams control. There are two alternatives:

* Use a string resource with Sql parameters as your command test (preferred)

### ****DO NOT: Include business logic in stored procedures****

The key reasons why we don't want to continue with this programming paradigm is maintainability, test-ability and scalability. The first because if a misbehavior is identified, there can be many tiers involved, the second because it would require that all our unit tests are have a database aside, the right data at the right place and after each non-idempotent unit test the database must be restored to the original state. None of these pre-conditions is easy to manage which is why data driven unit tests are not a widely used strategy. The scalability problem comes into the game because stored procedures scale only by database instance and not by service deployment. It is typically however much easier (and cheaper) to deploy a new micro service than adding a database into a cluster or working with database copies during cube calculations.

For this reason, all relevant logic has to happen inside the web-service package. Now, what is relevant?

* Non-trivial data transformations (date-time conversions, nvarchar to int, 'case' statements, truncations/ trims)
* All what requires a specification
* Any kind of computation

**It is typically a good indicator** to forget about the Sql engine while implementing the user story: if we use tomorrow MongoDb as our data store, would my service be still working? If the answer is not, more logic has to be moved into the service.

### ****DO NOT: use strings as command text****

**Bad example of a sql command**

|  |
| --- |
| "INSERT INTO Region (RegionID, RegionDescription) " +              "VALUES (@id, @desc)";          SqlParameter idParam = new SqlParameter("@id", SqlDbType.Int, 0);          SqlParameter descParam =              new SqlParameter("@desc", SqlDbType.Text, 100);          idParam.Value = 20;          descParam.Value = "First Region"; |

### AVOID: Stored Procedures in not owned databases

Reliable deployment is automated deployment. Automatically deploying and maintaining existing programmability in not owned databases is comparably complex to manage (creation, update, deletion) and each change exposes the risk that other applications are failing after removal (when they use our stored procedure). In order to eliminate this risk and problem we should use prepared sql statements in our applications (see [here](https://confluence.schroders.com/display/GCFN/SQL+Policies#SQLPolicies-SqlResourceText)).

### DO: Use string resource as command text

|  |
| --- |
| INSERT INTO Region (RegionID, RegionDescription) VALUES (@id, @desc) |

Sql\InsertRegion.sql

|  |
| --- |
| var command = new SqlCommandBuilder().FromResource("InsertRegion.Sql")                      .WithParameterValue("id", 20)                      .WithParameterValue("desc", "First Region").New(); |

Services\RegionService.cs

### DO: Include stored procedures into DACPACs

For owned databases, stored procedures (SP) should be part of the DACPAC packages so that they are deployed through Octopus automatically. The DACPAC must be written in such a way that it is idempotent (can be run several times on the same datasource and it never ends-up in an invalid state). This implies especially that the SP management must include:

* Creation of the script, if it doesn't exist
* Update of the script, if it does exist (can be drop & create)
* Removal of the script, if it doesn't exist any longer

# [SQL Policies](https://confluence.schroders.com/display/GCFN/SQL+Policies)

For the FundSphere project, the standard SQL policies and guidelines apply. The QA calendar can also be found here.

[Development Team QA Calendar.xlsx](https://confluence.schroders.com/download/attachments/29764133/Development%20Team%20QA%20Calendar.xlsx?version=2&modificationDate=1513268905457&api=v2)

Then however, in our commitment to service oriented design we need to limit artificially the responsibilities which may rely in stored procedures:

* [DO NOT: Include business logic in stored procedures](https://confluence.schroders.com/display/GCFN/SQL+Policies#SQLPolicies-DONOT:Includebusinesslogicinstoredprocedures)
* [DO NOT: use strings as command text](https://confluence.schroders.com/display/GCFN/SQL+Policies#SQLPolicies-DONOT:usestringsascommandtext)
* [AVOID: Stored Procedures in not owned databases](https://confluence.schroders.com/display/GCFN/SQL+Policies#SQLPolicies-AVOID:StoredProceduresinnotowneddatabases)
* [DO: Use string resource as command text](https://confluence.schroders.com/display/GCFN/SQL+Policies#SQLPolicies-SqlResourceTextDO:Usestringresourceascommandtext)
* [DO: Include stored procedures into DACPACs](https://confluence.schroders.com/display/GCFN/SQL+Policies#SQLPolicies-DO:IncludestoredproceduresintoDACPACs)

For deployment automation purposes, usage of stored procedures have to be reviewed completely - especially when the database is not under this teams control. There are two alternatives:

* Use a string resource with Sql parameters as your command test (preferred)

### ****DO NOT: Include business logic in stored procedures****

The key reasons why we don't want to continue with this programming paradigm is maintainability, test-ability and scalability. The first because if a misbehavior is identified, there can be many tiers involved, the second because it would require that all our unit tests are have a database aside, the right data at the right place and after each non-idempotent unit test the database must be restored to the original state. None of these pre-conditions is easy to manage which is why data driven unit tests are not a widely used strategy. The scalability problem comes into the game because stored procedures scale only by database instance and not by service deployment. It is typically however much easier (and cheaper) to deploy a new micro service than adding a database into a cluster or working with database copies during cube calculations.

For this reason, all relevant logic has to happen inside the web-service package. Now, what is relevant?

* Non-trivial data transformations (date-time conversions, nvarchar to int, 'case' statements, truncations/ trims)
* All what requires a specification
* Any kind of computation

**It is typically a good indicator** to forget about the Sql engine while implementing the user story: if we use tomorrow MongoDb as our data store, would my service be still working? If the answer is not, more logic has to be moved into the service.

### ****DO NOT: use strings as command text****

**Bad example of a sql command**

|  |
| --- |
| "INSERT INTO Region (RegionID, RegionDescription) " +              "VALUES (@id, @desc)";          SqlParameter idParam = new SqlParameter("@id", SqlDbType.Int, 0);          SqlParameter descParam =              new SqlParameter("@desc", SqlDbType.Text, 100);          idParam.Value = 20;          descParam.Value = "First Region"; |

### AVOID: Stored Procedures in not owned databases

Reliable deployment is automated deployment. Automatically deploying and maintaining existing programmability in not owned databases is comparably complex to manage (creation, update, deletion) and each change exposes the risk that other applications are failing after removal (when they use our stored procedure). In order to eliminate this risk and problem we should use prepared sql statements in our applications (see [here](https://confluence.schroders.com/display/GCFN/SQL+Policies#SQLPolicies-SqlResourceText)).

### DO: Use string resource as command text

|  |
| --- |
| INSERT INTO Region (RegionID, RegionDescription) VALUES (@id, @desc) |

Sql\InsertRegion.sql

|  |
| --- |
| var command = new SqlCommandBuilder().FromResource("InsertRegion.Sql")                      .WithParameterValue("id", 20)                      .WithParameterValue("desc", "First Region").New(); |

Services\RegionService.cs

### DO: Include stored procedures into DACPACs

For owned databases, stored procedures (SP) should be part of the DACPAC packages so that they are deployed through Octopus automatically. The DACPAC must be written in such a way that it is idempotent (can be run several times on the same datasource and it never ends-up in an invalid state). This implies especially that the SP management must include:

* Creation of the script, if it doesn't exist
* Update of the script, if it does exist (can be drop & create)
* Removal of the script, if it doesn't exist any longer

# [GIT Policies](https://confluence.schroders.com/display/GCFN/GIT+Policies)

GIT is the source control system selected by Schroders. For maintainability and quality assurance (QA) reasons it is evident that all moving parts, which are related to software or its deployment are under source control\*. The terminology used in this page is the following:

* [Definitions](https://confluence.schroders.com/display/GCFN/GIT+Policies#GITPolicies-Definitions)
* [Goals](https://confluence.schroders.com/display/GCFN/GIT+Policies#GITPolicies-Goals)
* [Continuous Integration Workflow](https://confluence.schroders.com/display/GCFN/GIT+Policies#GITPolicies-ContinuousIntegrationWorkflow)
* [Ignore List](https://confluence.schroders.com/display/GCFN/GIT+Policies#GITPolicies-IgnoreList)
* [Repository Naming Conventions](https://confluence.schroders.com/display/GCFN/GIT+Policies#GITPolicies-RepositoryNamingConventions)
* [Agreements](https://confluence.schroders.com/display/GCFN/GIT+Policies#GITPolicies-Agreements)
* [Branching Stategy](https://confluence.schroders.com/display/GCFN/GIT+Policies#GITPolicies-BranchingStategy)
* [Commit Process](https://confluence.schroders.com/display/GCFN/GIT+Policies#GITPolicies-CommitProcess)
* [Release Process](https://confluence.schroders.com/display/GCFN/GIT+Policies#GITPolicies-ReleaseProcess)
* [TBD: Defect Resolution Process](https://confluence.schroders.com/display/GCFN/GIT+Policies#GITPolicies-TBD:DefectResolutionProcess)

## Definitions

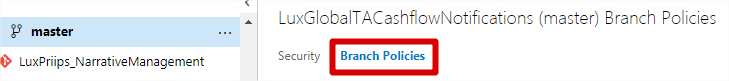
| **Word** | **Definition** |
| --- | --- |
| **Repository** | The self-contained collection of all software development artifacts including automated tests, deployment scripts and **their history**.  The master repository of the Cash Flow Notification project is: <http://tfs.schroders.com:8080/tfs/DefaultCollection/Schroders/Lux%20-%20Global%20TA%20Cashflow%20Notifications/_git/LuxGlobalTACashflowNotifications> |
| **Branches** | A branch represents a snapshot of the source at a given point in time inside a repository and is created through **forking**. Branches are not only essential to isolate different development paths from each other, but are also important for the ability to patch a release which is used in high-level environments (like production) without taking unnecessary risks. The branch of current development is known as **master**. |
| **Fork** | A variation of an already existing branch. A release branch is actually a fork of the master branch. All feature developments should happen on dedicated forks; this enables daily commits without impacting the team and allows also cheaper failure/ cancellation scenarios (work is stopped for the sprint). |
| **Clone** | The operation which creates a clone (on the work station) of a specified repository (all history, all branches, comments etc). Changes can be applied to it which then can be submitted (a.k.a. commit) to the branch on the server. This branch is also called **upstream**. |
| **Commit** | The process to apply changes to the upstream. Meaningful comments are strongly recommended. |
| **Pull-Request** | A request to integrate a change-set (of a fork) into the targeted branch, typically the master. |
| **Merge** | The operation which unifies two branches or a branch and a change-set. While simple divergences are solved by the repository software automatically, complex conflicts have to be solved by the developer. |

\* It is agreed that certain deployment scripts are embedded under the Octopus Deployment process. This is also good because Octopus also versions the deployment process per release.

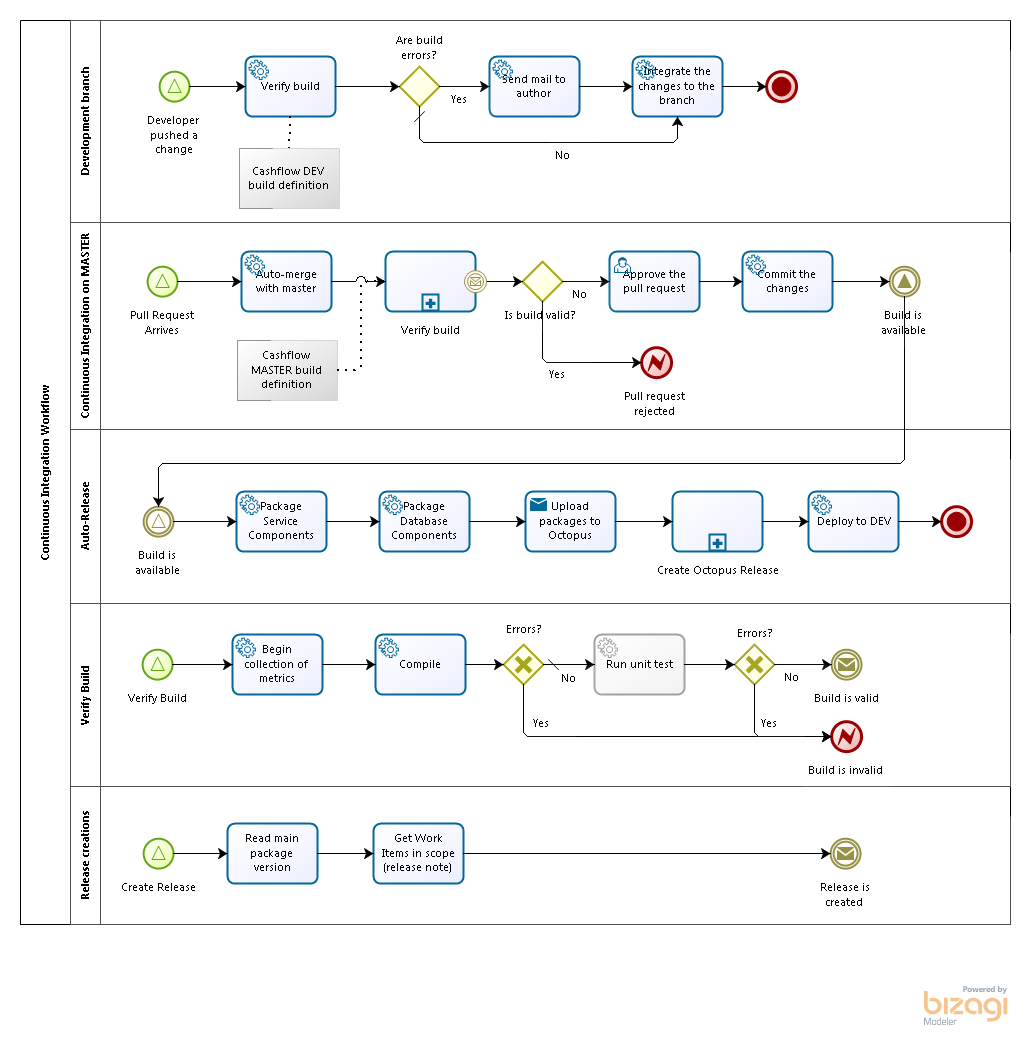
## Goals

The goals of these policies to establish a good balance between QA needs and development practices. Having this in mind, we committed to automate as many steps of the processes below (in **GREY**) as possible. Also, not all parts are valid currently and represent future goals. The reason for this is that first of all we need to deliver while establishing a good baseline. These steps will be enabled upon team agreement.

For the Cash Flow Notification project, the branch policies can be found only on the **master** branch:



## Continuous Integration Workflow



## Ignore List

Files which are build artifacts should not be in GIT because they pollute the change-sets and logs. For this reason, a .gitignore file should be added into each repository:

**.GitIgnore** Expand source

## Repository Naming Conventions

In order to harmonize the several projects under the TA capability, all GIT repositories need to start with:

|  |
| --- |
| Product.SimLux.TA --> for TA projects  Product.SimLux.CashFlow -- for CashFlow projects |

Further more, the **master branch must exist**and **represents the UAT version or newer** (it is never older). Additional development branches may exist.

## Agreements

* Each Work item has a dedicated branch from master in format: 'Work\_<pbi-number>'
* Only the master branch may be deployed to UAT
* PBI branches may be deployed to DEV/ DEV-INT/ SIT
* All pull-requests require at least 1 reviewer -the review should happen as soon as possible  a task for integration is recommended, assigned to the reviewer
* All pull-requests require at least 1 reviewer who is not the author
* All pull-requests are integrated into the master branch by the reviewer -upon successful review  a task for integration is recommended
* All commits contain meaningful and short summaries of the performed work
* ~~2 Pull-Request creation demands that the build compiles and all automatic tests succeed.~~

## Branching Stategy

The repository shall have:

* A **master** branch (default).
  + The branch is gated (must build, all unit tests must pass).
  + Builds of the branch go automatically published in Octopus.
* A **release** branch containing the released version (e.g. release-5.5).
  + The branch is gated (must build, all unit tests pass).
  + Builds of the branch get automatically published in Octopus.
* Additional**feature branches** may be created.
  + Feature branches follow the naming convention   
    **Work\_<PBI\_No>\_<Keyword>. Example: Work\_255466\_LoggingAndDashboard**
  + The branches should auto-build without gate.
  + Builds are not published automatically in Octopus.
  + The branch is erased after being intergrated into master and or release
* Additional**defect/ bug  branches** may be created.
  + Feature branches follow the naming convention <PBI>\_<Description.   
     **Bug\_<PBI\_No>\_<Keyword>. Example: Bug\_255446\_XRateWrongDate**
  + The branches should auto-build without gate.
  + Builds are not published automatically in Octopus.
  + The branch is erased after being intergrated into master and or release

 Notes on the **keyword**:

* The keyword should be **short**
* The keyword needs only to provide enough detail in order to distinguish the branch from other living branches.

## Commit Process

* Daily work has to be committed to the local repository and pushed to the branch.
* Each commit needs to be linked to at least one work item (enables automated release notes in Octopus).
* Each commit needs to highlight the current change (under the scope of the work item).

## Release Process

* Code used in production must be represented by a branch indicating the original version (e.g. a release branch)
* The aim is not to have more than 1 release branch to maintain at a time (but exceptions may apply)

## ~~TBD: Defect Resolution Process~~

* business continuity or not
* process for bugs during testing
* process for bugs on production

# [Application Versioning Scheme](https://confluence.schroders.com/display/GCFN/Application+Versioning+Scheme)

### Sprint Builds and Sprint Deliverables

The application version scheme is the usual <Major.Minor.Patch.Build>. In order to support efficient build, deployment and failure analysis it must be made sure that the version remains transparent across all steps of the SDLC. Sprint builds are kicked-off automatically on each commit to the master branch.

* The major build no should follow the PI number:
* The minor build no should follow the sprint number (within the PI)
* The patch number is an auto increment calculated during the build

<ProductIncrement>.<Sprint>.<BuildIncrement>

Example:  
**5.6.2** → PI 5 Sprint 6 Patch 2

TFS Build Definitions is [here](http://tfs.schroders.com:8080/tfs/DefaultCollection/Schroders/Lux%20-%20Global%20TA%20Cashflow%20notifications/_build?_a=completed&favDefinitionId=1420).

### Continuous Integration Builds

Continuous integration builds are kicked off regularly and aim to ensure that the software can be built and all automated tests are passing. They are normally **not intended** to be deployed because they may represent intermediate work which can corrupt existing instances, if deployed.

**0.**<ProductIncrement>.<BuildIncrement>

Example:  
**0.6.113** → PI 6 Build 113

TFS Build Definitions is [here](http://tfs.schroders.com:8080/tfs/DefaultCollection/Schroders/Lux%20-%20Global%20TA%20Cashflow%20notifications/_build?_a=completed&favDefinitionId=1548).

### Release Builds

Release builds are the only ones which are supposed to be deployed to PP and PROD. They need to be kicked-off manually so that the changelog contains all work items since the last build (=last release). The version scheme for release builds is the following:

<ProductIncrement>.<Sprint>\*10.<BuildIncrement>:

Example:  
6.**40**.7  → PI 6, Sprint 4 Patch 7

TFS Build Definitions is not available yet.

 Please note that release builds can only be done from release branches (e.g. release-x.y)

## Further Requirements

* Build Number must be unique and assigned systematically to all published builds  
  → Avoids human errors  
  → Avoids duplicates  
  → Application version is consistent across all modules of the build package
* Non-Final Builds must have diverging version, for instance 0.1.x. Put differently only builds from master can have a build number >1.0  
  → Allows to see if an intermediate build has been deployed to an environment
* Build Number must be known to the application and printed to the logs  
  → Allows to map an error to an application version. Especially useful for non-development environments
* Release Number in Octopus should reflect the build number  
  → Allows to know directly from the Dashboard, which build is actually deployed
* Release Number should contain the PI Increment and Sprint (Major.Minor)  
  → This allows to guesstimate the delta between the master branch and the production.  
  → It is immediately visible, whether the PI final is deployed or a sprint deliverable (x.6 indicates last sprint of the PI, a value <6 must be a sprint deliverable)

# [REST Guidelines](https://confluence.schroders.com/display/GCFN/REST+Guidelines)

The REST service interfaces we intend to host are either meant to be used by consumers outside of the Service Fabric or for testing purposes. REST is a great protocol, but unfortunately, it provides a lot of potential for implementation ambiguity leading to an inconsistent API. Please note that API in here refers to the public service API.

The complete REST guidelines can be found on GitHub: '[Microsoft API guidelines](https://github.com/Microsoft/api-guidelines)', further referred to as 'Microsoft Document'

* [URL Structure](https://confluence.schroders.com/display/GCFN/REST+Guidelines#RESTGuidelines-URLStructure)
* [Canonical Identifier](https://confluence.schroders.com/display/GCFN/REST+Guidelines#RESTGuidelines-CanonicalIdentifier)
* [Supported Methods](https://confluence.schroders.com/display/GCFN/REST+Guidelines#RESTGuidelines-SupportedMethods)
* [Response Formats](https://confluence.schroders.com/display/GCFN/REST+Guidelines#RESTGuidelines-ResponseFormats)
* [Item Keys](https://confluence.schroders.com/display/GCFN/REST+Guidelines#RESTGuidelines-ItemKeys)
* [Serialization](https://confluence.schroders.com/display/GCFN/REST+Guidelines#RESTGuidelines-Serialization)
* [Collection URL Patterns](https://confluence.schroders.com/display/GCFN/REST+Guidelines#RESTGuidelines-CollectionURLPatterns)
* [Nested Collections and Properties](https://confluence.schroders.com/display/GCFN/REST+Guidelines#RESTGuidelines-NestedCollectionsandProperties)
* [Service Reponses](https://confluence.schroders.com/display/GCFN/REST+Guidelines#RESTGuidelines-ServiceReponses)

## URL Structure

Humans SHOULD be able to easily read and construct URLs.

This facilitates discovery and eases adoption on platforms without a well-supported client library.

An example of a well-structured URL is:

|  |
| --- |
| <https://api.contoso.com/v1.0/people/jdoe@contoso.com/inbox> |

An example URL that is not friendly is:

|  |
| --- |
| https://api.contoso.com/v1.0/items?url=<https://resources.contoso.com/shoes/fancy> |

(source: chapter 7.1 of the Microsoft document)

## Canonical Identifier

In addition to friendly URLs, resources that can be moved or be renamed SHOULD expose a URL that contains a unique stable identifier. It MAY be necessary to interact with the service to obtain a stable URL from the friendly name for the resource, as in the case of the "/my" shortcut used by some services.

The stable identifier is not required to be a GUID.

An example of a URL containing a canonical identifier is:

|  |
| --- |
| https://api.contoso.com/v1.0/people/7011042402/inbox |

(source: chapter 7.3 of the Microsoft document)

## Supported Methods

Operations MUST use the proper HTTP methods whenever possible, and operation idempotency MUST be respected. HTTP methods are frequently referred to as the HTTP verbs. The terms are synonymous in this context, however the HTTP specification uses the term method.

Below is a list of methods that Microsoft REST services SHOULD support. Not all resources will support all methods, but all resources using the methods below MUST conform to their usage.

| **Method** | **Description** | **Is Idempotent** |
| --- | --- | --- |
| GET | Return the current value of an object | True |
| PUT | Replace an object, or create a named object, when applicable | True |
| DELETE | Delete an object | True |
| POST | Create a new object based on the data provided, or submit a command | False |
| HEAD | Return metadata of an object for a GET response. Resources that support the GET method MAY support the HEAD method as well | True |
| PATCH | Apply a partial update to an object | False |
| OPTIONS | Get information about a request; see below for details. | True |

(source: chapter 7.4 of the Microsoft document)

It very common mistake **not** to use the verbs for standard actions and use custom actions instead. Examples:

I want to...

...delete all foreign exchanges...  
**BAD**: **GET**./globalCashFlows/api/1.0/foreignExchanges**/deleteAll  
GOOD**: **DELETE**./globalCashFlows/api/1.0/foreignExchanges

...see all exchange rates for USD... **BAD**: **GET**./globalCashFlows/api/1.0/loaders/foreignExchanges**/USD/load  
GOOD**: **GET**./globalCashFlows/api/1.0/loaders/foreignExchanges**/USD**

...load all exchange rates into FundSphere... **BAD: GET./globalCashFlows/api/1.0/loaders/foreignExchanges/USD/load  
GOOD: POST**./globalCashFlows/api/1.0/loaders/foreignExchanges/loadAll

## Response Formats

For organizations to have a successful platform, they must serve data in formats developers are accustomed to using, and in consistent ways that allow developers to handle responses with common code.

Web-based communication, especially when a mobile or other low-bandwidth client is involved, has moved quickly in the direction of JSON for a variety of reasons, including its tendency to be lighter weight and its ease of consumption with JavaScript-based clients.

JSON property names SHOULD be camelCased.

Services SHOULD provide JSON as the default encoding.

(source: chapter 7.10 of the Microsoft document)

## Item Keys

Services MAY support durable identifiers for each item in the collection, and that identifier SHOULD be represented in JSON as "id". These durable identifiers are often used as item keys.

Collections that support durable identifiers MAY support delta queries.

(source: 9.1 of the Microsoft document)

## Serialization

Collections are represented in JSON using standard array notation.

(source: 9.2 of the Microsoft document)

## Collection URL Patterns

Collections are located directly under the service root when they are top level, or as a segment under another resource when scoped to that resource.

For example:

|  |
| --- |
| GET https://api.contoso.com/v1.0/people |

Whenever possible, services MUST support the "/" pattern. For example:

|  |
| --- |
| GET https://{serviceRoot}/{collection}/{id} |

Where:

* {serviceRoot} – the combination of host (site URL) + the root path to the service
* {collection} – the name of the collection, unabbreviated, pluralized
* {id} – the value of the unique id property. When using the "/" pattern this MUST be the raw string/number/guid value with no quoting but properly escaped to fit in a URL segment.

(source: 9.3 of the Microsoft document)

## Nested Collections and Properties

Collection items MAY contain other collections. For example, a user collection MAY contain user resources that have multiple addresses:

|  |
| --- |
| GET https://api.contoso.com/v1.0/people/123/addresses |
| {    "value": [      { "street": "1st Avenue", "city": "Seattle" },      { "street": "124th Ave NE", "city": "Redmond" }    ]  } |

(source: 9.3.1 of the Microsoft document)

## Service Reponses

(These are REST standard responses)

| **Status** | **Definition** |
| --- | --- |
| **200** | Service operation successful, all mappings available |
| **200** | Service operation successful, some or all mappings unknown (service completed, but not all mappings are available |
| **401** | Could not authenticate the requester |
| **403** | Authentication succeed, but requester is not allowed to access the resource |
| **4XX** | Bad request: request is not properly formatted or not understood In body: description of the error (see error object) |
| **5XX** | Service faulted; could not fulfill the request In body: description of the error (see error object) |

# [Confluence Guidelines](https://confluence.schroders.com/display/GCFN/Confluence+Guidelines)

This chapter collects guidelines on how for this project Confluence is supposed to be used in order to remain a good source of reliable information.

* [DO: Back-specify changes applied in the scope of a Product Backlog Item](https://confluence.schroders.com/display/GCFN/Confluence+Guidelines#ConfluenceGuidelines-DO:Back-specifychangesappliedinthescopeofaProductBacklogItem)
* [DO NOT: Specify (mapping) details on a confluence page and in an attachment](https://confluence.schroders.com/display/GCFN/Confluence+Guidelines#ConfluenceGuidelines-DONOT:Specify(mapping)detailsonaconfluencepageandinanattachment)

### DO: Back-specify changes applied in the scope of a Product Backlog Item

A product backlog item, also known as story represents a change request which should be completed in the scope of a sprint. Confluence pages however are normally not linked to sprints, they either represent the AS-IS and/or the TO-BE situation. There are cases however in which the backlog item has no backing Confluence content, for instance, it changes:

* details of an existing feature
* introduces a new feature which has no footprint yet in Confluence

In these cases, the change should be described also in Confluence, see [the definition of done of the PBI](https://confluence.schroders.com/display/GCFN/Definitions+Of+Done).

### DO NOT: Specify (mapping) details on a confluence page and in an attachment

Consistency is key and showing information on a page and attach the same information in a file (Word, Excel) implies that both are equally maintained. This is unlikely the case for which reason the duplication should be avoided:

1. Either the information is only on the page
2. Or the information is only in the attached file

The first fits better internal development flows, second suits better in cases where the information gathering and maintenance is with third parties.

# [Octopus Policies](https://confluence.schroders.com/display/GCFN/Octopus+Policies)

To every release package in Octopus, relevant build information should be in. This includes:

* ~~Related work items of the published build~~
* ~~The origin of the build~~
* ~~Populate builds with the environments in which it has been deployed~~

(only marked boxes are implemented)

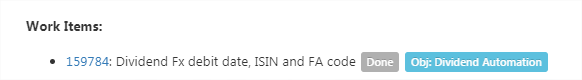
### Naming Convention

We agreed to align the Octopus naming convention to the TA naming conventions:

|  |
| --- |
| Product Global CashFlow -- for upcoming CashFlow projects  Product Global CashFlow Fabric - Service Fabric  Product Global CashFlow Databases - Databases (Staging and Export) |

### Show the related work items of a build

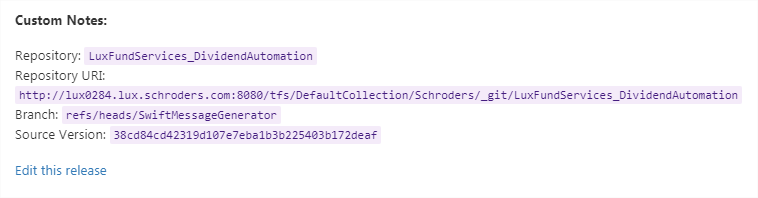
Since Octopus is the deployment tool to use and we want to avoid costly compilation of release notes, we should link our commits towards **product backlog items** so that they can appear in the release note visible in Octopus.



TFS #180071

### Show Technical Release Details in Octopus

Showing technical release details is particularly important when one application co-exists in multiple versions on several environments. In such cases, all log entries should be in one or another way linked to the origin of the application's code base.



TFS #180072

### Mark Builds in when released with Octopus

Having the perspective of a TFS Build list, it is impossible to known which build(s) have been released where and when. Octopus can channel this back  through a particular step inside the deployment process.