DRAFT - IT Project Guidance

Design: Technical – Common Interface Patterns

Version:

0.1

## Description

<TODO>

## Synopsis

A small number of common interfaces improve abstraction, improve reuse, improve certainty, and reduce testing effort required.

## Contents

[Description 1](#_Toc145232973)

[Synopsis 1](#_Toc145232974)

[Contents 2](#_Toc145232975)

[Synopsis 3](#_Toc145232976)

[Introduction 3](#_Toc145232977)

[Heading Level 3 3](#_Toc145232978)

[Heading Level 4 3](#_Toc145232979)

[Appendices 4](#_Toc145232980)

[Appendix A - Document Information 4](#_Toc145232981)

[Images 4](#_Toc145232982)

[Tables 4](#_Toc145232983)

[References 4](#_Toc145232984)

[Review Distribution 4](#_Toc145232985)

[Audience 4](#_Toc145232986)

[Structure 4](#_Toc145232987)

[Diagrams 4](#_Toc145232988)

[Terms 5](#_Toc145232989)

## Introduction

Some typed languages permit the development of contracts that define how contract implementations are to be defined. For example, In C#, any [class](https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/class) or [struct](https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/struct) definition that implements that contract must provide an implementation of the members defined in the interface.

Contracts improve the modularity and adaptability and testability of systems while reducing brittleness, by permitting the application of Liskov’s Substitution Principle.

Certain common patterns occur repeatedly in and across systems. These recuring patterns are perfect candidates for being defined as interfaces.

This document outlines a common set of such contracts to consider implementing.

## Areas

Although not fixed or finite, some general ‘areas’ can be discerned from the types of interfaces introduced in this document:

* **Configuration**: steps involved in the integration and configuration of the system during its initial startup.
* **Services**: the stateless infrastructure, business and application specific implementations of functionality used to manipulate models.
* **Models**: the shape of logical models managed by the system, whether they are infrastructural, or business related. These in turn can be classified as either:
  + **System models** – the entity and value objects that are kept *internal* to the system, which Services work with, and are persisted with the help of a Repository Service (wrapping an ORM).
  + **DTO models** – the entity and value objects exposed to outside the system by integration and/or UX APIs schemas.  
    Note that for maintainability and security reasons DTO models that are exposed externally should *always* be different and not reuse internal logic models.

Attributes that are common to find on models address:

* + Reference Id or Key,
  + Auditability,
  + State,
  + Title and Description,
  + Key,
  + Value: many items have a value, whether it be float based (eg: Task % Done), money based (eg: Price), integer based (e.g., Count).
  + User Ratings and Feedback: feedback is an asset used to improve the value of systems, one model at a time.
  + **Default Rendering instructions**: most models are displayed at one point, hence may have default display styling and/or icon that needs to be associated with the item, etc.
* Options: a often recurrent type of model are Option List Items, which allow system and users to specify one or more options.
* Rendering:
* Systems are organised as
  + Configuration Steps used to configure a system,
  + stateless instances of Services that develop, modify, and return Models,
  + Models are developed from assembly of one or more attributes from a small set of oft-repeated ones (Ids, State, Title, Description, etc.etc.)
  + ObjectMapping Definitions.
* Ids are almost always UUIDs, and for distribution and performance reasons developed on the server side (not within databases).
* Options are design to allow user addition and modification, with safeguards to disable the removing of required system defaults.
* Options are designed to be informative to end users with Titles and Descriptions, thereby displayable in many more options than just WinForm Select dropdowns.
* Options are designed so as to be definable and organisable beforehand, to be brought online in the future.

## Naming Conventions

The default convention is to name Interface definitions with an ‘I’ prefix.

Alternatively, a convention is to name Interface definitions with an ‘IHas’ prefix.

Note:  
I currently can’t remember when I started using the second prefix.   
There is a reason, somewhere.   
It may have to do with searching for ‘IHas’ being more effective than searching for ‘I’.

# Interfaces

### Data Store Identifiers

Datastore Identifiers are the Ids used by the datastore/database.

#### IHasId<T>

Db identifiers in trivial systems are usually integers, but in most systems – especially ones that permit offline operations – use UUIDs. Hence the property has to be a generic.

Interface IHadId<TId> where TId:scale {

TId Id {get;set;}  
}

#### IHasUUId

In distributed systems – which most services that use one or more horizontally scaled devices are – Identifiers must be UUIDs, hence every single Id we use unless there is a very compelling reason are based on the following:

Interface IHasUUId : IHasId<Guid> {}

Note:  
It is very important to note that when UUIDs are used as identifiers we never rely on database manager created random UUIDs -- being random, they trash clustered indexes, causing large scale negative impact to performance.   
  
Instead, we develop “Time-based prefix+random suffix” based UUIDs in the logic tier when developing new models. The UUID based IDs that are generated this way are sequential (as they rely on datetime) with enough randomness to reasonably reduce the chance of items created at the exact same millisecond on different devices having the same Id.

### Enabled

The following contracts are used to specify an object’s (service or model) state.

Examples of Enabling something is

#### IHasIsEnabled

The following contract is probably one of the simplest.   
It’s used to define system Users, etc.

Interface IHasIsEnabled {

bool Enabled {get;set;}  
}

Note:  
The value can and is recommended to be *dynamic*, based on the values from IHasIsEnabledFromUtc & IHasIsEnabledToUtc, described next.

#### IHasIsEnabledFromToUtc

While IHasIsEnabled is simple enough, and useful to shut out a system User immediately – for example - it does not have sufficient attributes to enable setting up things ahead of time, for later use.   
  
An example would be setting up Groups of Users (e.g., next semester’s learners in a classroom), without enabling them just yet.

interface IHasIsEnabledFromToUtc : IHasEnabled {

DateTime FromDateTimeUtc {get;set;}  
 Nullable<DateTime> ToDateTimeUtc {get;set;}  
}

Note:  
This interface can be applied directly to an object -- in which case only one time slot in one context can be defined – in which case multiple timeslots can be defined, across multiple contexts (e.g.: tenancies).

### State

State management is more or less the essence of information management systems – they manage Resources through a workflow of states.

States vary from system to system and use case to use case.

#### IHasState

In trivial systems, it is just a Boolean value:

Interface IHasState {

bool State {get;set;}  
}

A Boolean state is usually insufficient in business systems over the long run.

They require custom logic to render, but also mature over to time such that the value can be any one of a list of small Options.

#### IHasState<T>

Interface IHasState<T> where T:IHasOption {

T State {get;set;}  
}

Note:  
It should point to use-specific list of Reference Data/Code Set options that are generally one or more of the following states:

Creating, Reviewing, Rejected, Approved, Published, Replaced, Merged, Removed, Archived, Restored.

### Options

System, User Settings usually are based on choosing an option from a list.

Interface IHasOption : IHasGuidId, IHasKeyAndValue, IHasTitleAndDescription, IHasSystemOptionValue {

}

The above contract is a composite of *IHasUUId* (described earlier) and simple contracts described below.

#### IHasKey

Keys and Names (see *IHasName* elsewhere) are not interchangeable. An Key is immutable textual identifier and something the system’s logic may refer to, whereas a Name is a mutable label, viewed and maybe even referenced by end users.

Interface IHasKey {

Get Key {get;set;}

}

#### IHasValue<T>

A key is often associated to a value, the type depending on the use case.

Interface IHasValue<T> {

<T> Value {get;set;}

}

Note:  
If it makes things easier later to not have to specify generic types every time, one can define type specific derivatives (IHasIntValue, IHasDoubleValue, IHasFloatValue, IHasMoneyValue), but it is not essential.

#### IHasKeyAndValue<T>

As stated before an immutable Key is generally associated to a Value

Interface IHasKeyAndValue :IHasName, IHasKey<T> {}

The Key being immutable, a key/value Option may (probably always, somewhere) also have a displayable Title and Description.

Note:  
The Key and Title of an Option is only rarely the same value (eg: only when the default language is en-US, and the default language for development is also en-US).

#### IsSystemOption

If forsaking the use of enums, and some Options are to be edited by end users, there is value in stopping some values from being deleted as the rest of the system relies on them.

Interface IsSystemOption {

Bool IsSystem {get;set;}  
}

### Displayed Option Attributes

While some values are used internally and not shown to end users (e.g.: Key/Values) other options are end user input and are expected to be displayed for selection in an approachable, transparent manner.

#### IHasDisplayTitleAndDescription

Coming from a background highly influenced by Windows development dropdowns and options, and later the same from HTML, it seemed relatively obvious that drop downs just needed a visible Title. I was wrong. Better designed user interfaces are far richer, and are able to display a list of options with a title and description, as well as a icon of something.

Interface IHasTitleAndDescription {

string Title {get;set;}

string Description {get;set;}

}

#### IHasNaturalDisplayOrder

The original/natural order of an item in a list can be an attribute of an option.

Interface IHasNaturalDisplayOrder {

Integer DisplayOrder {get;set;}

}

Note:  
There are also good arguments for the display order & style being externailised in a wrapping/joining object so that these properties, which are really specific to the rendering of the option -- not the business purpose of the option itself. That’s correct – but it appears to be a relatively high amount of work only to decouple it for purity reasons.

#### IHasNaturalDisplayStyle

The primary user interface protocol is HTML, which permits styling objects with a CSS class:

Interface IHasNaturalDisplayStyle {

string DisplayStyleId {get;set;}

}

Note:  
See note on IHasNaturalDisplayOrder.

### Configuration and Settings

Configuration and Settings are not the same thing. Configuration is set (e.g.: via a config.xml file) before a system starts and remains immutable over the during runtime whereas Settings are mutable system and user preferences that are persisted in a operational data store (e.g.: Db), and *can* be changed during runtime.

#### IHasConfigurationStep

At the start-up of systems, their infrastructure integrations must be configured.

To discover these steps by reflection/discovery one can inherit their interfaces from a common base instance. This permits listing the configuration that took place, and the success thereof

It also provides a way to title the step and describe its purpose, as well as turn it off or on as the case may be.

Interface IHasConfigurationStep : IHasTitleAndDescription, IHasEnabled, IHasOutcome {

}

interface IHasOutcome {

Double PercentCompleted get;

State LastResult get;

RecordState(State state, double percent);  
}

### IHasService

Domain Driven Development (DDD) relies on the system following Object Oriented (OO) based development methodologies. One aspect of OO is the use of instanced (as opposed to singleton) Services.   
To facilitate finding them by Reflection/Auto-discovery, Service contracts should inherit from *IHasService*.

The contract offers no attributes or functionality.

Note:  
There are competing reasons to not do so, and instead find services by convention (e.g., classes that end with “Service”, for example), in order to discover Services in assemblies that are not under your development control. But in such cases, you would create a in-app wrapping class -- inheriting IHasService -- to in turn instantiate and invoke the 3rd party service class.

Interface IHasService {

}

### Auditings

Auditing of record changes is a fundamental part of being able to apply accountability.

The following attributes are helpful with auditing and are intended to be applied to entitities related to a system entities.

#### IHasCreatedByAndWhenUtc<TUserId>

Interface IHasCreatedByAndWhenUtc {

TUserId CreatedUserId {get;set;}

DateTime CreatedWhen {get;set;}

}

#### IHasModifiedByAndWhenUtc<TUserId>

Interface IHasModifiedByAndWhenUtc : IHasCreatedByAndWhenUtc {

TUserId ModifiedUserId {get;set;}

DateTime ModifiedWhen {get;set;}

}

#### IHasDeletedByAndWhenUtc<TUserId>

Interface IHasDeletedByAndWhenUtc : IHasModifiedByAndWhenUtc {

TUserId DeletedUserId {get;set;}

DateTime DeletedWhen {get;set;}

}

Note:  
In almost all circumstances physical deletion is not recommended. If you want to delete a resource or anything else, logically delete it by setting its State to ‘Removed’ or similar. The benefits of logical deletes are numerous and the flexibility they provide far outweigh the simplicity of development of physical deletion -- including meeting archiving needs, permitting Undos, removing flow-stopping modal confirmation dialogs, reducing training requirements, etc.

Interface IHasArchivedByAndWhenUtc : IHasModofiedByAndWhenUtc {

String ArchivedUserId {get;set;}

DateTime ArchivedWhen {get;set;}

}

Notice that auditing a reference to a User Id supports the design pattern that User Records are *never* physically deleted, just anonymized -- or they in turn lead to either relational errors or cascading deletes of all records they were involved with, including only Modified.

#### IHasTrackingInformation

Interface IHasTrackingInformation : IHasDeletedByAndWhenUtc, IHasArchivedByAndWhenUtc { }

The values have nothing to do with the business purpose of the entity, so there is a strong argument to making them part of an associated collection of editing events:

Interface IHasTrackingInformation<T> where T: IHasTrackingInformation {

ICollection<IHasTrackingInformation> TrackingInformation {get;set}  
}

With a collection in place, the ORM save event is overridden, and new objects are added to the collection just before the save operation takes place.

### IHasObjectMappingDefinition

One configuration step is to search and find by reflection/auto discovery instances of object-to-object map definitions. These are used to map, at the API layer, internal system models to integration Data Transfer Objects (DTOs) and back again.

Interface IHasObjectMappingDefinition<TSystemModel,TDTO>{

TDTO Map(TSystemModel model);

TSystemModel(TDTO dto);   
}

## Ratings and Feedback

Is User feedback part of a system, or part of a supporting feedback system? Either way it should track who made the rating, so should inherit from *IHasAuditability* while being able to removed from a listing if inappropriate.

Interface IHasRating : IHasUUId, IsEnabled, IHasAuditability {

Int Rating {get;set;}

String Comment {get;set;}   
}

Again, this has nothing to do with the purpose of the entity being rated, so should be external to it, part of a collection of ratings (each with Auditability of course):

Interface IHasRatings {

Collection<THasRating> IHasRating;  
}

Appendices

Appendix A - Document Information

### Authors & Collaborators

Sky Sigal, Solution Architect

### Versions

* 1. Initial Draft

### Images

[Figure 1: TODO Image 2](#_Toc144995112)

### Tables

[Table 1: TODO Table 3](#_Toc145048484)

[Table 2: TODO Table 2 3](#_Toc145048485)

### References

**There are no sources in the current document.**

### Review Distribution

The document was distributed for review as below:

|  |  |
| --- | --- |
| Identity | Notes |
|  |  |
|  |  |
|  |  |

### Audience

The document is technical in nature, but parts are expected to be read and/or validated by a non-technical audience.

### Structure

Where possible, the document structure is guided by either ISO-\* standards or best practice.

### Diagrams

Diagrams are developed for a wide audience. Unless specifically for a technical audience, where the use of industry standard diagram types (ArchiMate, UML, C4), is appropriate, diagrams are developed as simple “box & line” monochrome diagrams.

### Terms

Refer to the project’s Glossary.

##### IT

: acronym for Information, using Technology to automate and facilitate its management.

##### ICT

: acronym for Information & Communication Technology, the domain of defining Information elements and using technology to automate their communication between entities. IT is a subset of ICT.