Delta Capability Framework

Version 1.0

Author: Jasbir S. Minhas

Table of Contents

[1) Introduction 3](#_Toc497830623)

[2) Major Components 4](#_Toc497830624)

[A. Configuration XML and XML Schema Definition 4](#_Toc497830625)

[B. Framework Major Group of Classes 5](#_Toc497830626)

[I. CapabilityManager Group of Classes 6](#_Toc497830627)

[II. CapabilityKey Group of Classes 7](#_Toc497830628)

[3) Framework JUnit classes 9](#_Toc497830629)

[4) Framework code coverage and code violations 11](#_Toc497830630)

[5) Framework integration with exiting Delta Code 11](#_Toc497830631)

# Introduction

Delta Capability Framework is designed to make the merged Delta codebase to be declaratively configurable for different jurisdictions and to address future configuration needs. It has the following features:

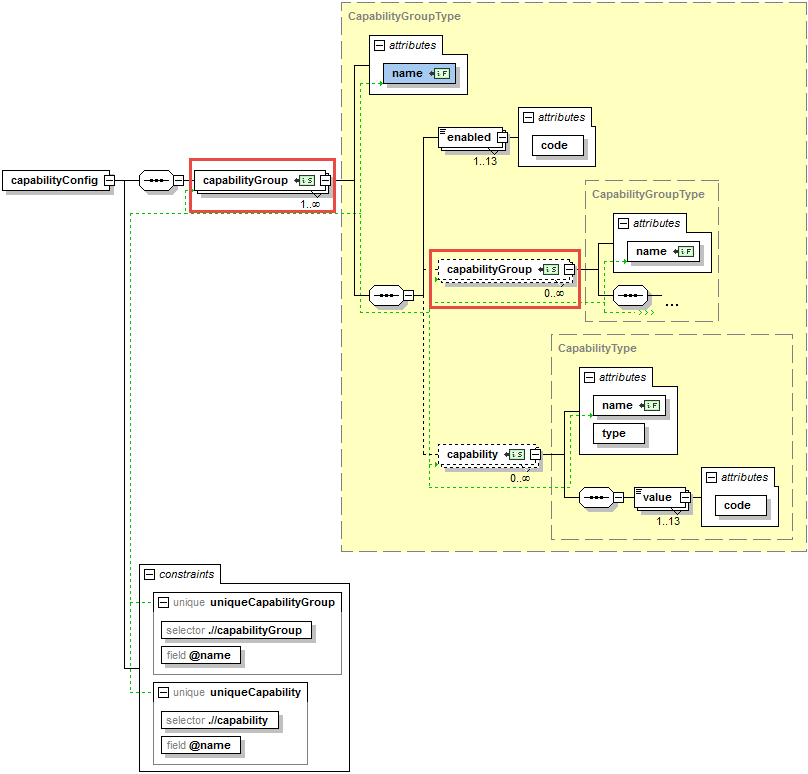
* XML / Hierarchical based configuration for various capability
* Ability to group the capabilities and declaratively turn on and off the groups
* Supporting XML Schema Definition to capture configuration issues during development and runtime
* Includes JUnit test cases cover all permutations of configuration
* Includes current configuration for NS, NB and ON.
* Includes adapter so that current Delta code can use this framework without too many changes to the existing code. One enum and one class method needs to be changed and I have provided the implementations for those as well.

# Major Components

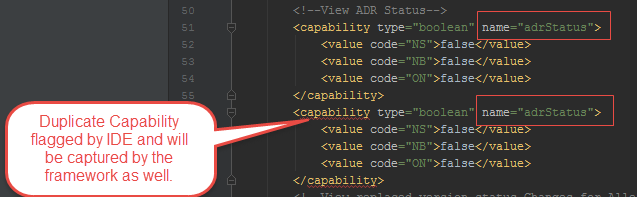
## Configuration XML and XML Schema Definition

The Capability Configuration XML is a hierarchical structure designed to capture configuration data of all provinces in Capability Groups.

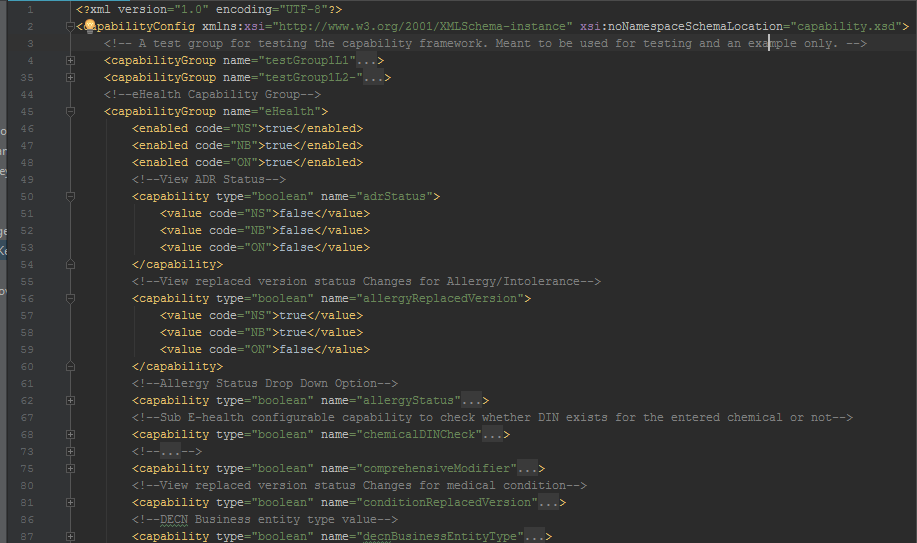
The following XML Schema Definition (XSD) diagram depicts various components of this structure. Capabilities can be organized in various groups and the framework allows switching these capabilities declaratively on or off at a capability level or at a group level without changing the framework code. The schema definition and the framework support recursive capability groups in a single artifact.



The XSD defines various constraints to help validate the configuration XML at development time, through IDEs or XML editor, and at application boot time though the Capability Framework. For example a duplicate Capability Group or a duplicate Capability will be highlighted by the XML/IDE as shown in the following screen shot.



Following XML snippet shows definition of eHealth capabilities.



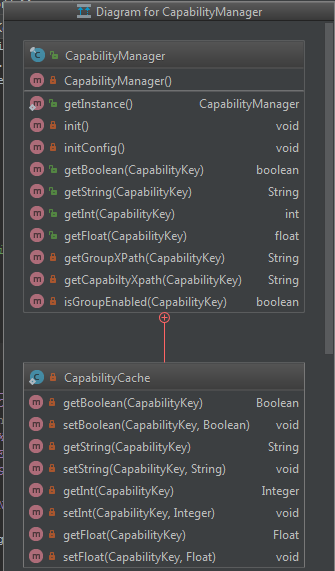
## Framework Major Group of Classes

The Capability Framework has the following two major group of classes.

### CapabilityManager Group of Classes

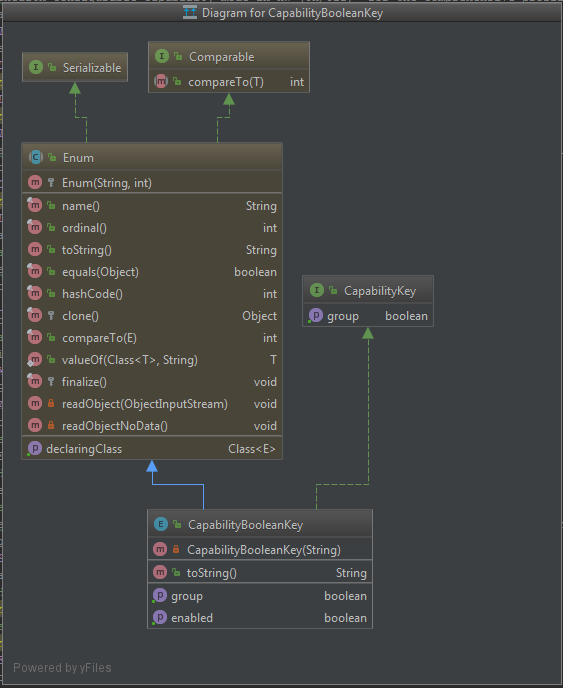
This is designed to be a singleton class and has the following features:

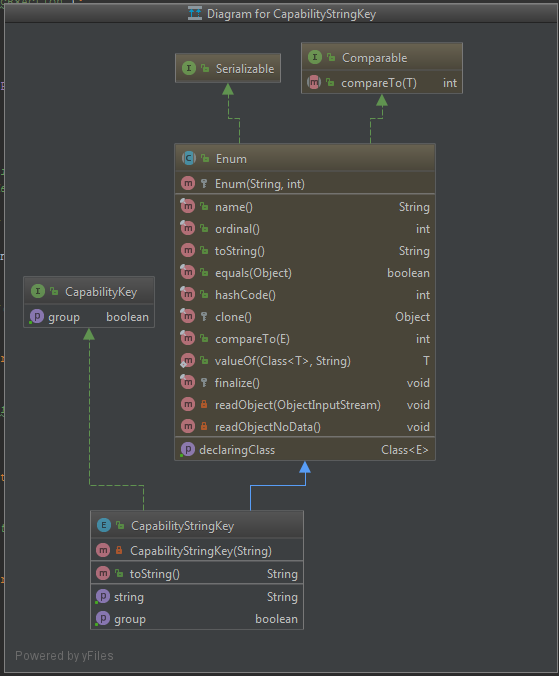
* Loads, parses and validates the Capability configuration against Capability configuration XML schema definition
* Dynamically builds XPath based on a simple expression to identify each capability
* Provides strongly typed interfaces to avoid bugs and any misuse of the framework
* Handles infinite (limited by memory) number of capability groups and sub-groups
* It has an inner static class called CapabilityCache which is used to cache various capabilities as they are utilized by the application to improve the performance. Each capability is cached as they are utilized first time.



### CapabilityKey Group of Classes

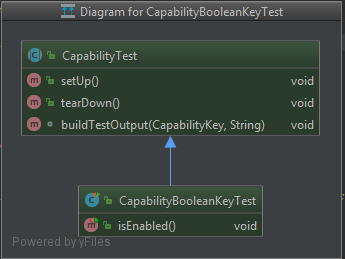
The framework support four types of strongly typed capabilities; Boolean, Integer, Float and String to avoid bugs. The keys for these capabilities are Enums and they implement the CapabilityKey interface. This interface declares the common method required by CapabilityManager. Due to Java 7 limitation, which is fixed in Java 8, the default method can’t be implemented in the CapabilityKey interface and hence implanted in individual key subclasses. Since the key subclasses are of type Enums which are implicitly inherited from java.lang.Enum and Java does not allow multiple inheritance we are force to implement CapabilityKey as a Java interface rather than an abstract class. A “TODO:” comments for the developers have been made in the code to move the implementation of the common method from individual key classes into the CapabilityKey interface once the Delta application is upgraded to Java 8. The following class diagrams depicts the class hierarchy of the two CapabilityKey subclasses.

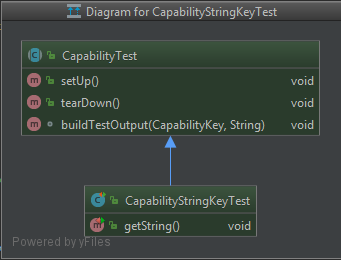




# Framework JUnit classes

JUnit classes have been created to test various permutations of Capability configuration. They classes dynamically test every capability. Following class diagrams show two of these test classes.





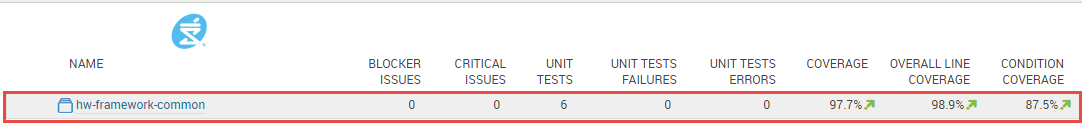
Junit Test cases

* listBooleans
* listStrings
* testBoolean
* testDeletedConfigAtStart
* testDeletedConfigMidExecution
* testDouble
* testIllFormedConfigMidExec
* testIllFormednessConfigAtStart
* testInt
* testInvalidConfigAtStart
* testInvalidConfigMidExec
* testMisplacedConfigAtStart
* testMisplacedConfigMidExecution
* testString

# Framework code coverage and code violations

The following screen depicts that:

* there are zero Blockers or Critical issues in Capability Framework
* there is 98.9% overall line coverage



# Framework integration with exiting Delta Code

The following changes needs to be made to integrate the Capability Framework with the existing Delta Code.

* Replace com.sdm.hw.store.dto.SubEHealthConfigConstants enum from PWF (completed and will be provide with the framework code)
* Replace isSubEHealthEnabled method of the com.sdm.hw.store.services.HwStoreServiceImpl class with the implementation provided as part of the package (completed and will be provide with the framework code)
* Implement getCurrentProvinceCode method of com.sdm.hw.common.capability. ProvinceCodeProvider singleton class. Delta 17.1 team needs to replace the current implantation of this method with code which does the following:
  + Read current province code from the Store\_preference table
  + Return corresponding com.sdm.hw.common.capability.ProvinceCode enum