

# Mobile Application Store CollectionServiceAPI Design Document

Date: 2013-10-16

Author: David Killeffer <rayden7@gmail.com>

Reviewer(s): Jonathan Nichols <jn42887@gmail.com>

## Introduction

This document defines the design for the Mobile Application Store Collections Service API, which is a core component of the Mobile Application Store. This document is organized into the following sections:

1. **Overview:** summarizes the problem that is being solved and why
2. **Summary of Product API Changes:** explains changes that were required to the ProductAPI to accommodate refactorings based on TA feedback from Assignment 2
3. **Requirements:** enumerates the various requirements for the CollectionServiceAPI
4. **Use Cases:** enumerates the use cases that the CollectionServiceAPI must support
5. **Class Diagram:** graph of the primary classes and their relationships that are in use in the CollectionServiceAPI
6. **Sequence Diagram:** shows the primary relationship between StaticCollections/DynamicCollections and how they relate to the ProductAPI
7. **Class Dictionary:** a catalog of the classes that collectively comprise the CollectionServiceAPI and the various methods, properties, and attributes that define each
8. **Implementation Details:** special considerations and explanatory observations about how the classes relate to each other and work
9. **Changes from Original Design & Requirements:** explanations of how this implementation varies slightly from the stated requirements in MobileApplicationStoreCollectionServiceRequirements.pdf
10. **Testing:** explanation of how the classes in CollectionServiceAPI are to be tested, and how the TestDriver exercises the functionality
11. **Risks:** identifies and enumerates any potential pitfalls or deficiencies in the current implementation and issues that might arise from this implementation
12. **Instructions for Compiling and Running Application:** explains how to compile and run the application

## Overview

The various Content items that live in the ProductAPI live as individual islands with no conceptual relationship to each other; this makes it difficult to present groups of related Content items to consumers, and also makes browsing the product catalog difficult. The Collection Service API allows the organization of Products into Collections. By organizing Products into Collections, related groups of products can be grouped by category, or organized into taxonomical structures. These new collections of content will be very useful

for presenting groups of Content items to consumers so that they can easily browse related content by theme, category, price, etc.

Collections can be static lists of individual Product items, or dynamic (e.g., the Content items that comprise a dynamic collection are based on a ContentSearch criteria object that is run against the ProductAPI).

## Summary of ProductAPI Changes

Several refinements and refactorings were performed to the ProductAPI for the CollectionServiceAPI, but these were mainly small functionality additions, but also some structural changes.

The first functional change was to add a method to the ProductAPI that will allow for retrieving Content items based on their contentID. The ProductAPI already had methods for retrieving countries and devices in a similar fashion (***getCountryByCode(String:code)*** and ***getDeviceByID(String:deviceId)***), so really this change was both for the sake of consistency (e.g., all publicly-accessible “things” that the ProductAPI is responsible for should be able to be quickly and easily retrieved by their ID), but also for the CollectionServiceAPI to be able to more easily retrieve Content items. The CollectionServiceAPI has a soft requirement to “wrap” Content items in a proxy object, and so being able to easily retrieve Content items by ID made this a requirement. The new method added to ProductAPI is:

```
public Content getContentByID(String:contentID);
```

There was some large refactorings done to the cscie97.asn3.ecommerce.exception package; all exceptions for the entire application (both ProductAPI and CollectionServiceAPI) were combined under this single umbrella. Additionally, new exception types were added. A new custom super-type exception was added called “MobileAppException”; this exception is basically the same as the old ImportException (has attributes for lineWhereFailed, lineNumberWhereFailed, filename, originalCause, etc.). Since most of the custom exceptions all shared the same properties, the new MobileAppException is declared to be the abstract exception type parent class for the ImportException, ContentNotFoundException, and CollectionNotFoundException classes. This keeps the custom exception types leaner and also limits code repetition (in keeping with D.R.Y. principles).

The last large refactoring of the ProductAPI comes in the cscie97.asn3.ecommerce.csv package. Since the CollectionServiceAPI needs to load collections via CSV files, define collection attributes and add items to collections, and also to search collections, there was some refactoring that needed to take place. Since DynamicCollections have a ContentSearch attribute, and in the old ProductAPI the only location where such items were being constructed was in the ***SearchEngine.executeQuery(String:query)*** method, the ***executeQuery()*** method was refactored to be broken out into two parts:

1. first, take the CSV query line and parse it out, constructing a ContentSearch object (created a new method called ***getContentSearchForCSV(String:query)*** on the

SearchEngine; now both the ProductAPI and the CollectionServiceAPI can make use of that method and benefit from the easy creation of ContentSearch objects)

2. take the ContentSearch object, and pass it to the ProductAPI to run the actual content search

By breaking the method up into two smaller methods, the utility of the SearchEngine class was enhanced by adding a new client (the CollectionServiceAPI), and also the longer-term maintainability of the class enhanced since now methods have smaller, more limited sets of responsibility.

## CollectionServiceAPI Requirements

This section defines the requirements for the CollectionServiceAPI system component. Given that future sprints will incorporate authentication and authorization (Authentication Service) and that the CollectionServiceAPI will need to interact with the Authentication Service, the design accounts for the notion of **protecting restricted interface** use cases, and will use a mock authentication token for now (for example, see requirements under “Creating Collections” and “Adding Content to Collections” in *MobileApplicationStoreCollectionServiceRequirements.pdf*).

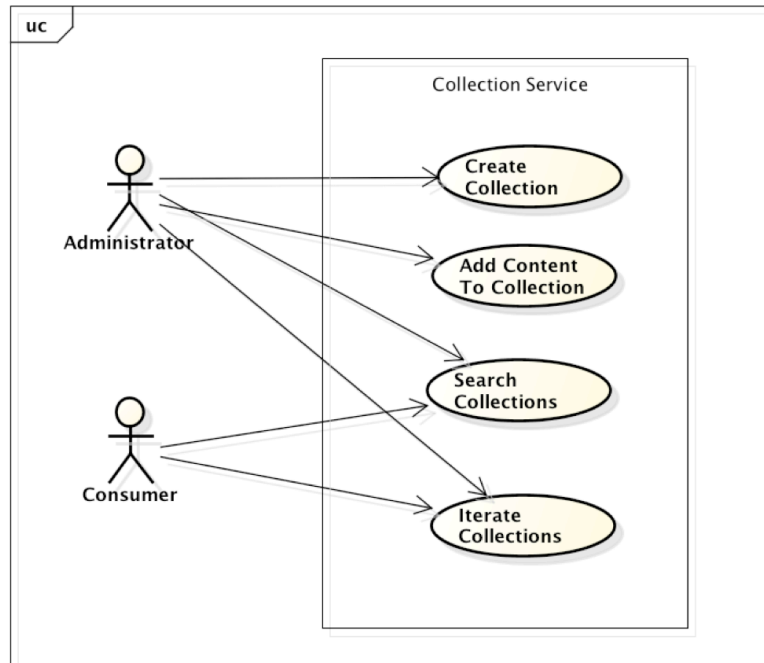
Administrators and Application Developers that have a GUID access token should be able to create new Collections (whether Static or Dynamic), and both add discrete Content (via ContentProxy objects that wrap Content items) to StaticCollections and define the search criteria for DynamicCollections. Collections may be nested and contain child Collections, making a Collection an arbitrarily complex tree structure, and granting Administrators and Application Developers the ability to create organized taxonomies of Content items. For DynamicCollections, their primary children should be the result of executing a predefined ContentSearch (the same type as already built in the ProductAPI). Adding content to StaticCollections, defining the search criteria on DynamicCollections, and adding child Collections to existing Collections are similarly **restricted interfaces**, and all require a valid GUID access token.

Consumers, Administrators, and Application Developers should be able to search Collections and find ones that match supplied search criteria on the Collection name or description. Since Collections may contain child Collections and child Content items (in the form of ContentProxy objects), Consumers, Administrators, and Application developers must be able to iterate over all the children of Collections.

More specific details on the requirements are found in the *MobileApplicationStoreCollectionServiceRequirements.pdf* document.

## Use Cases

The following use case diagram shows the major functional tasks that each type of user of the Collection Service can perform.



**Create Collection:** Administrators can create and define new Collections. These collections can be Static (e.g., consist of discrete, pre-existing Content items from the ProductAPI) or Dynamic (contain a ContentSearch property, that, when executed, returns a list of Content items). The content items in a Collection are wrapped as ContentProxy objects so that they share similar properties to the Collections themselves and can be iterated over as a common object. This is a restricted interface and requires a validated GUID token (mocked for now).

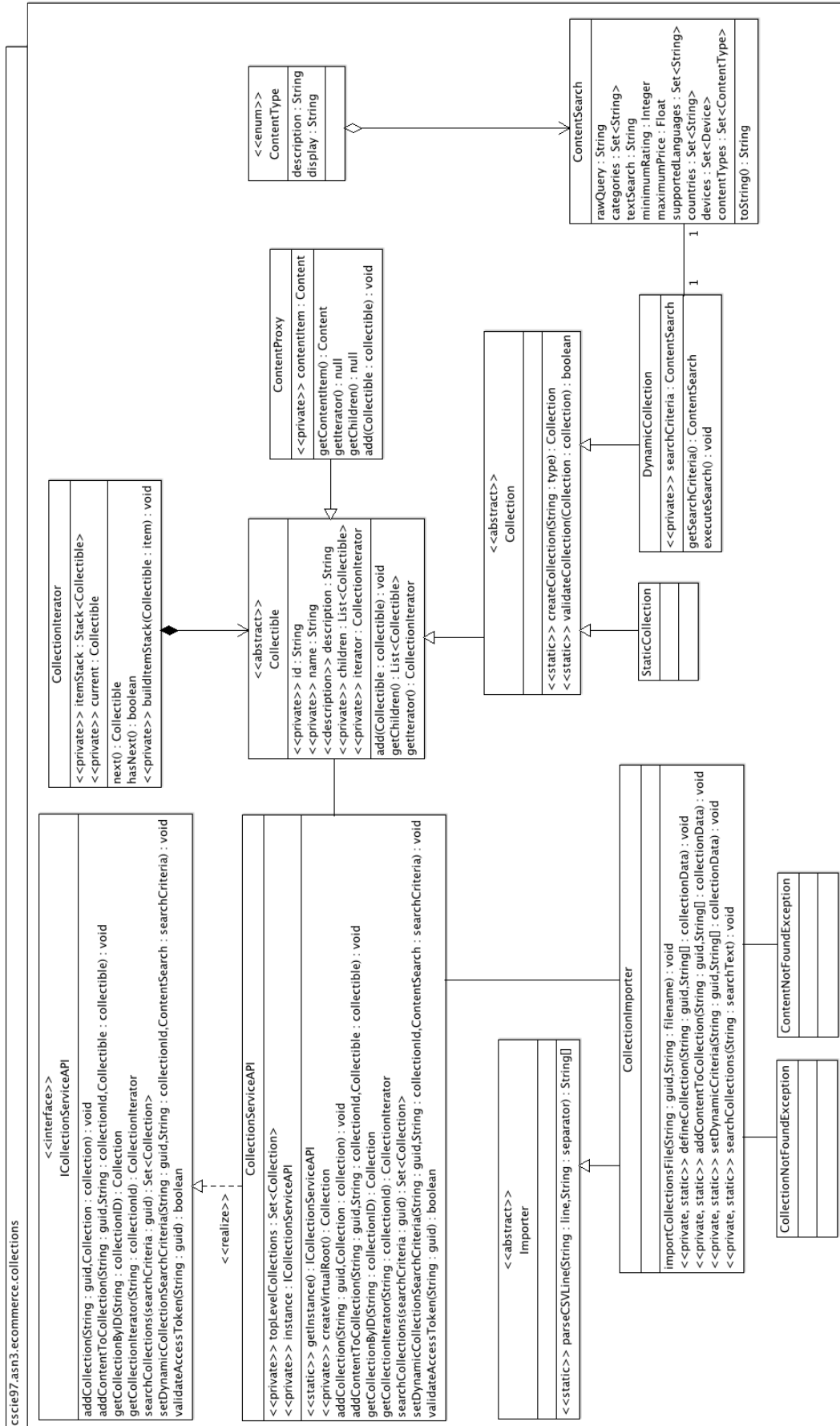
**Add Content To Collection:** Administrators can add content to StaticCollections and define the search criteria for DynamicCollections. This is a restricted interface and requires a validated GUID token (mocked for now). Collections may also be added to Collections, making a Collection an arbitrarily complex tree structure.

**Search Collections:** both Administrators and Consumers may search the Collection catalog for Collections that match the search text in either the collection's name or description, and get a set of matching Collections. In order to search for collections that match the search criteria, this use case has an implicit dependency on the *Iterate Collections* use case.

**Iterate Collections:** both Administrators and Consumers may iterate over all the Collections in the catalog. Since both the Collections themselves and the ContentProxy items that they may contain inherit from the same abstract class, the iterator should allow for both Collections and the Content they contain to be enumerated by the iterator. This should allow for quick traversal of either a limited set of Collections or the entire catalog of Collections at once.

## Class Diagram

The following class diagram defines the major classes defined in this design.



**ICollectionServiceAPI / CollectionServiceAPI:** The primary class for Consumers and Administrators to interact with the Collection catalog is through the CollectionServiceAPI class. ICollectionServiceAPI provides an interface to define the methods that the service must support (programming to interfaces rather than concrete classes is a good general design principle).

**CollectionImporter:** this is the primary mechanic by which Collections are created and imported into the catalog, content is added to Collections, content search criteria are defined on DynamicCollections, iterate over Collections, and search for matching Collections. The CollectionImporter itself only exercises the functionality of the CollectionServiceAPI. The CollectionImporter is called via the TestDriver, which loads CSV files for operation; a single CSV file called collections.csv is the primary way that all major functionality is exercised. The CollectionImporter may throw customized exceptions CollectionNotFoundException ContentNotFoundException when attempting to look up collections or content; it also uses Importer for parsing the CSV file.

**Collection:** provides a factory method for creating “blank” StaticCollections or DynamicCollections based on the type passed, and also provides a static method for validating Collection objects once they have had the minimal set of required attributes defined (id, name, description). Collection is also the abstract parent class of both StaticCollection and DynamicCollection. Collection also extends Collectible, where it gets most of it’s functionality from.

**StaticCollection:** primarily a marker class, since the functionality of static collections is defined in it’s parent Collection and Collectible abstract classes.

**DynamicCollection:** allows for a Collection whose child elements are returned as a result of executing a predefined ContentSearch (exactly the same ContentSearch as is defined in the ProductAPI). Once the ContentSearch property of a DynamicCollection is defined (whether at object creation or afterwards), the search criteria is immediately executed against the ProductAPI so that the children are defined as the return result of the executed search.

**Collectible:** abstract parent class to Collection and ContentProxy, defining the major properties (id, name, description), as well as the children and iterator. Collectible items may be iterated over and contain many levels of child Collectibles. The Collectible uses the Composite pattern to define what “collections” are; they are groups of ContentProxy items and also StaticCollection and DynamicCollection objects; so ContentProxy, StaticCollection, and DynamicCollection objects are all instances of Collectible. Collectible objects can be iterated over by use of their CollectionIterator property.

**CollectionIterator:** used to simplify the arbitrarily complex nature of Collections, which can contain ContentProxy items, as well as StaticCollection and/or DynamicCollection objects. Since Collections are complex trees, the iterator uses a stack to add Collectibles as it traverses the Collection and then uses that stack to traverse the collection. Iteration is run in a depth-first traversal.

## Sequence Diagram

This sequence diagram shows how a Customer might interact with the CollectionServiceAPI to iterate over the items in a Collection.

//TODO: create and add  
Sequence Diagram here

## Class Dictionary

This section specifies the class dictionary for the CollectionServiceAPI. The primary classes should be defined within the package “cscie97.asn3.ecommerce.collection”. Per the changes outlined previously under the “Summary of ProductAPI Changes” section, exception classes should be defined within the package “cscie97.asn3.ecommerce.exception”, and CSV-handling classes should be defined within the package “cscie97.asn3.ecommerce.csv”.

Package: **cscie97.asn3.ecommerce.collection**

### *ICollectionServiceAPI* <<interface>>

This interface defines the service contract for any classes that intend to implement these methods and act as the concrete implementation of the CollectionServiceAPI. Public interface for the CollectionServiceAPI. Consumers and Administrators may search over all collections that match searchText in the Collection name or description. Administrators may also create collections, add content to collections, and for DynamicCollections, define the search criteria used.

### *Methods*

Method Name	Signature	Description
addCollection	(String : guid, Collection : collection) : void	Restricted interface; will validate GUID token before adding content to a collection. Adds the passed collection to the CollectionService catalog

		at the top-level.
addContentToCollection	(String : guid, String : collectionId, Collectible : collectible) : void	Restricted interface; will validate GUID token before adding content to a collection. Looks up the Collection with matching collectionId in the catalog and then adds the passed Collectible to that Collection. Note that Collectibles may be either ContentProxy items (which wrap Content items that are returned by the IProductAPI, or Collection objects.
getCollectionById	(String : collectionID) : Collection	Given a collection ID, search for any Collection that matches that code in the collection catalog. Constructs a private virtual "root" collection and sets all current top-level collections as it's children so that it can iterate over every collection in the catalog to find the one that matches.
getCollectionIterator	(String : collectionId) : CollectionIterator	If collectionID passed is not null and corresponds to a valid collection, simply returns the iterator for that Collection (note that the CollectionIterator can also simply be retrieved if a reference to the actual Collection instance exists). If the passed collectionId is null or empty string, constructs a virtual "root" level collection that has all the current top-level Collections as children, and returns a CollectionIterator which will be able to iterate over every item in the Collection catalog.
searchCollections	(String : searchCriteria) : Set<Collection>	<p>Finds all Collections whose Collection name or description contains any part of the searchCriteria passed. Note that the search is case-insensitive.</p> <p>To conduct the search, constructs a virtual "root" Collection that has all the top-level Collections as immediate children, and then iterates over all the Collectibles in this virtual Collection aggregate and finds matching Collections.</p>
setDynamicCollectionSearchCriteria	(String : guid, String : collectionId, ContentSearch : searchCriteria) : void	Restricted interface; will validate GUID token before adding content to a collection. Looks up the Collection with matching collectionId in the catalog, ensures that the found collection is actually a DynamicCollection, and then sets the ContentSearch searchCriteria object on the DynamicCollection. Note that at the time the search criteria on a DynamicCollection is defined, it is immediately executed so that the child elements of the DynamicCollection are present.
validateAccessToken	(String : guid) : Boolean	Verifies that the guid access token passed is authenticated and authorized for carrying out restricted actions on the CollectionServiceAPI (such as adding new Collections, adding Content to Collection, etc.).



		Note that for this version of the CollectionServiceAPI, this method is mocked and will return true for any string passed.
--	--	---

## ***CollectionServiceAPI***

Implements the ICollectionServiceAPI class to provide public methods for interaction with the Collection catalog. Consumers and Administrators may search over all collections that match searchText in the Collection name or description. Administrators may also create collections, add content to collections, and for DynamicCollections, define the search criteria used.

The CollectionServiceAPI is accessed as a Singleton; users must use the CollectionServiceAPI.getInstance() method to obtain a reference to the sole CollectionServiceAPI instance.

## ***Methods***

Method Name	Signature	Description
addCollection	(String : guid, Collection : collection) : void	Restricted interface; will validate GUID token before adding content to a collection. Adds the passed collection to the CollectionService catalog at the top-level.
addContentToCollection	(String : guid, String : collectionId, Collectible : collectible) : void	Restricted interface; will validate GUID token before adding content to a collection. Looks up the Collection with matching collectionId in the catalog and then adds the passed Collectible to that Collection. Note that Collectibles may be either ContentProxy items (which wrap Content items that are returned by the IProductAPI, or Collection objects.
getCollectionById	(String : collectionID) : Collection	Given a collection ID, search for any Collection that matches that code in the collection catalog. Constructs a private virtual "root" collection and sets all current top-level collections as it's children so that it can iterate over every collection in the catalog to find the one that matches.
getCollectionIterator	(String : collectionId) : CollectionIterator	If collectionID passed is not null and corresponds to a valid collection, simply returns the iterator for that Collection (note that the CollectionIterator can also simply be retrieved if a reference to the actual Collection instance exists). If the passed collectionId is null or empty string, constructs a virtual "root" level collection that has all the current top-level Collections as children, and returns a CollectionIterator which will be able to iterate

		over every item in the Collection catalog.
searchCollections	(String : searchCriteria) : Set<Collection>	<p>Finds all Collections whose Collection name or description contains any part of the searchCriteria passed. Note that the search is case-insensitive.</p> <p>To conduct the search, constructs a virtual "root" Collection that has all the top-level Collections as immediate children, and then iterates over all the Collectibles in this virtual Collection aggregate and finds matching Collections.</p>
setDynamicCollectionSearchCriteria	(String : guid, String : collectionId, ContentSearch : searchCriteria) : void	<p>Restricted interface; will validate GUID token before adding content to a collection. Looks up the Collection with matching collectionId in the catalog, ensures that the found collection is actually a DynamicCollection, and then sets the ContentSearch searchCriteria object on the DynamicCollection. Note that at the time the search criteria on a DynamicCollection is defined, it is immediately executed so that the child elements of the DynamicCollection are present.</p>
validateAccessToken	(String : guid) : Boolean	<p>Verifies that the guid access token passed is authenticated and authorized for carrying out restricted actions on the CollectionServiceAPI (such as adding new Collections, adding Content to Collection, etc.).</p> <p>Note that for this version of the CollectionServiceAPI, this method is mocked and will return true for any string passed.</p>
getInstance	() : ICollectionServiceAPI	<p>Returns a reference to the single static instance of the CollectionServiceAPI, and will construct a new instance if one does not exist already (follows Singleton pattern). In keeping with the generally accepted programming practice of coding to interfaces rather than concrete classes, the return value is declared to be an interface type.</p>
createVirtualRoot	() : Collection	<p>To iterate over all the collections in the CollectionServiceAPI catalog, a "virtual" root collection must be created, and all current topLevelCollections added to it. The virtual root Collection will then be able to iterate over every single Collection and Content item in the entire Collection catalog. This method will construct that virtual root Collection, add all the top-level Collections to it, and then returns the virtual root.</p>

## Properties

Property Name	Type	Description
topLevelCollections	Set<Collection>	The unique top-level collections contained in the Collection catalog; each collection may only be declared at the top-level once, but may be nested arbitrarily deeply in other collections. Required.
instance	ICollectionServiceAPI <<static>>	Singleton instance of the ICollectionServiceAPI. Required.

## Collectible <<abstract>>

Abstract parent class for Collection and ContentProxy objects. Collectibles can contain child Collectibles, which makes them a tree structure. Because arbitrary numbers of child Collectibles can be added, the class supports a CollectionIterator for iterating over all child elements. The object types that may be created that are children of the Collectible class are:

- StaticCollection
- DynamicCollection
- ContentProxy

All of these types will inherit their shared attributes (Collectible.id, Collectible.name, Collectible.description) from this class.

## Methods

Method Name	Signature	Description
getIterator	() : CollectionIterator	Returns the iterator for the current collection. The iterator also follows the Singleton pattern; once the iterator has been declared and initialized, the already-declared one will be returned. If the iterator has not been defined when getIterator() is called, a new CollectionIterator will be created and initialized.
getChildren	() : List<Collectible>	Returns the current set of children Collectible objects that are contained by this Collectible. Note that while the children of the current Collectibles are all themselves Collectible objects, but their <i>actual</i> types may be one of: <ul style="list-style-type: none"><li>• ContentProxy (used to wrap the Content objects that get returned by the IProductAPI)</li><li>• StaticCollection</li><li>• DynamicCollection</li></ul> Note that if the current Collectible is actually a DynamicCollection this method will still return the found ContentProxy objects that are found by executing the search (no need to separately execute the search).
add	(Collectible : collectible) : void	Adds a child Collectible to the current Collectible. Since the children of the current Collectible will be

		modified as a result, a side effect of adding a new Collectible is to null out the current iterator (so that the next time a client wishes to iterate over the Collectible the new item will be included).
toString <<override>>	() : String	Overrides generic toString() method and print out all the properties of a Collectible. Useful for debugging.

### *Properties*

Property Name	Type	Description
id	String	Unique identifier for the Collectible. Required.
name	String	Name of the Collectible. Required.
description	String	Textual description of what this Collectible is. Required.
children	List<Collectible>	All children Collectibles of the current Collectible; this makes the Collectible a tree structure.
iterator	CollectionIterator	Used for iterating over this Collectible and all children elements.

## Application

Models an application content item; extends Content.

### Methods

Method Name	Signature	Description
validateContent <<static>>	(Content content) : Boolean	Given an Application content item, validate that the attributes follow the required guidelines for an Application content item. Calls Content.validateContent() to validate the basic properties common to all content items.
toString <<override>>	() : String	Overrides generic toString() method and print out all the properties of an Application content item. Useful for debugging.
equals <<override>>	(Object : compare) : Boolean	Overrides the default equals method to compare the passed Application content item to the current one. If all properties of the two objects match completely, returns true; false otherwise. Used since Application items may be stored in the ProductAPI in collections, and objects in Sets need to be compared for equality.
hashCode	() : Integer	Overrides the default hashCode method to generate a unique code for the current object. Similar to equals(), overridden to ensure that object comparisons use the properties of the object.

### Properties

Property Name	Type	Description
fileSizeBytes	Integer	The number of bytes the application is on disk. Required.

## Ringtone

Models a ringtone content item; extends Content.

### Methods

Method Name	Signature	Description
validateContent <<static>>	(Content content) : Boolean	Given a Ringtone content item, validate that the attributes follow the required guidelines for a Ringtone content item. Calls Content.validateContent() to validate the basic properties common to all content items.
toString <<override>>	() : String	Overrides generic toString() method and print out all the properties of a Ringtone content item. Useful for debugging.
equals <<override>>	(Object : compare) : Boolean	Overrides the default equals method to compare the passed Ringtone content item to the current one. If all properties of the two objects match completely, returns true; false otherwise. Used since Ringtone items may be stored in the ProductAPI in collections, and objects in Sets need to be compared for equality.
hashCode	() : Integer	Overrides the default hashCode method to generate a unique code for the current object. Similar to equals(), overridden to ensure that object comparisons use the properties of the object.

### Properties

Property Name	Type	Description
durationInSeconds	Float	How many seconds in length the ringtone is. Required.

## Walpaper

Models a wallpaper content item; extends Content.

### Methods

Method Name	Signature	Description
validateContent <<static>>	(Content content) : Boolean	Given a Wallpaper content item, validate that the attributes follow the required guidelines for a Wallpaper content item. Calls Content.validateContent() to validate the basic properties common to all content items.
toString <<override>>	() : String	Overrides generic toString() method and print out all the properties of a Wallpaper content item. Useful for debugging.
equals <<override>>	(Object : compare) : Boolean	Overrides the default equals method to compare the passed Wallpaper content item to the current one. If all properties of the two objects match completely, returns true; false otherwise. Used since Wallpaper items may be stored in the ProductAPI in collections, and objects in Sets need to be compared for equality.
hashCode	() : Integer	Overrides the default hashCode method to generate a unique code for the current object. Similar to equals(), overridden to ensure that object

		comparisons use the properties of the object.
--	--	---

### **Properties**

Property Name	Type	Description
pixelWidth	Integer	How tall the wallpaper is in pixels. Required.
pixelHeight	Integer	How tall the wallpaper is in pixels. Required.

### **Device**

Models a device that content items can be downloaded to.

### **Methods**

Method Name	Signature	Description
validateDevice <<static>>	(Device : device) : Boolean	Given a Device, validate that the attributes follow the required guidelines for a Device. Called when importing new Devices into the product catalog to ensure that only valid Devices get added.
toString <<override>>	() : String	Overrides generic toString() method and print out all the properties of a Device. Useful for debugging.
equals <<override>>	(Object : compare) : Boolean	Overrides the default equals method to compare the passed Device to the current one. If all properties of the two objects match completely, returns true; false otherwise. Used since Device items may be stored in the ProductAPI in collections, and objects in Sets need to be compared for equality.
hashCode	() : Integer	Overrides the default hashCode method to generate a unique code for the current object. Similar to equals(), overridden to ensure that object comparisons use the properties of the object.

### **Properties**

Property Name	Type	Description
id	String	The unique identifier for the device. Required.
name	String	Name of the device. Required.
manufacturer	String	Name of the manufacturer of the device. Required.

### **Country**

Models a unique country where content may or may not be allowed to be downloaded to devices connected in this country.

### **Methods**

Method Name	Signature	Description
validateCountry <<static>>	(Country : country) : Boolean	Given a Country, validate that the attributes follow the required guidelines for a Country. Called when importing new Countries into the product catalog to ensure that only valid Countries get added.
toString <<override>>	() : String	Overrides generic toString() method and print out all the properties of a Country. Useful for debugging.
equals <<override>>	(Object : compare) : Boolean	Overrides the default equals method to compare the passed Country to the current one. If all properties of the two objects match completely, returns true; false otherwise. Used since Country items may be stored in the ProductAPI in collections, and objects in Sets need to be compared for equality.
hashCode	() : Integer	Overrides the default hashCode method to generate a unique code for the current object. Similar to equals(), overridden to ensure that object comparisons use the properties of the object.

### *Properties*

Property Name	Type	Description
code	String	Unique 2-character country code. Required.
name	String	Name of the country. Required.
exportStatus	String	Export status of the country. Can be either “open” or “closed”. Required.

## ContentType

Marker class for the unique types of discrete content supported by the Mobile Application Store. Current valid types for ContentType objects are APPLICATION, RINGTONE, and WALLPAPER. When creating an Application, Ringtone, or Wallpaper instance a ContentType matching the object’s type is created and set on the content item object.

### *Properties*

Property Name	Type	Description
description	String	1 or 2 sentence description of what the enum item is.
display	String	Short, 1-word description of what the enum item is.

## ContentSearch

Captures the attributes specified in a content item search so that a single object can be passed from SearchEngine to ProductAPI, and also in the future could be more easily logged or reported on. Note that the ContentSearch does not execute any search logic itself, it merely contains the attributes to search for; the ProductAPI bears the responsibility for



executing the ContentSearch by comparing its properties to everything in the product catalog.

### Methods

Method Name	Signature	Description
toString <<override>>	() : String	Overrides generic toString() method and print out all the properties of a ContentSearch item. Useful for debugging.

### Properties

Property Name	Type	Description
rawQuery	String	Contains the CSV line comprising the original search query sent to SearchEngine.
categories	Set<String>	Any content categories to match on any existing content items in the Product catalog on.
textSearch	String	Basic text to search across all content items for. This text will be searched across the content item name, description, and authorName attributes of the content item.
minimumRating	Integer	The rating of content to search for on a 0 to 5 scale where 5 is best. Default value is 6 so that this parameter will not match any content items unless the search criteria specifically sets the value to something between 0-5 (inclusive).
maximumPrice	Float	The maximum price to search for content items on. Defaults to -1 so that a basic search where this value isn't set won't automatically match everything in the catalog.
supportedLanguages	Set<String>	The list of supported languages to search content items for. Note that partial string matches are supported (e.g., if a supportedLanguage code has a string "fr", and there is a content item with language code "fr_CA", the content item will match).
countries	Set<Country>	List of countries to match content items on.
Devices	Set<Device>	List of devices to match content items on.
contentTypes	Set<ContentType>	List of ContentTypes to match content items on.

### Importer

Used to load Devices, Countries, and Content items (Applications, Wallpapers, and Ringtones) into the Product catalog.

### Methods

Method Name	Signature	Description
parseCSVLine <<protected>>	(String : line, String : separator) : String[]	Utility method used by both Importer and SearchEngine to handle parsing CSV lines in all the import files (countries.csv, devices.csv, and products.csv). Takes a line with comma

		separated values and parses it into a String array.
importCountryFile <<static>>	(String : guid, String : filename) : void : throws ImportException, ParseException	Imports a CSV file and parses each line, attempting to add a new Country to the Product catalog for each line. Throws ImportException or ParseException for file handling issues or formatting problems with the CSV. <i>Restricted Interface - must supply a GUID token to import content [currently mocked; all strings accepted]</i>
importDeviceFile <<static>>	(String : guid, String : filename) : void : throws ImportException, ParseException	Imports a CSV file and parses each line, attempting to add a new Device to the Product catalog for each line. Throws ImportException or ParseException for file handling issues or formatting problems with the CSV. <i>Restricted Interface - must supply a GUID token to import content [currently mocked; all strings accepted]</i>
importContentFile <<static>>	(String : guid, String : filename) : void : throws ImportException, ParseException	Imports a CSV file and parses each line, attempting to add a new Content item (Application, Ringtone, or Wallpaper) to the Product catalog for each line. Throws ImportException or ParseException for file handling issues or formatting problems with the CSV. <i>Restricted Interface - must supply a GUID token to import content [currently mocked; all strings accepted]</i>

## SearchEngine

Used to execute content item queries from the Product catalog. Parses CSV file with specified search criteria on each line.

### Methods

Method Name	Signature	Description
executeQuery <<static>>	(String : queryLine) : void : throws ParseException	Imports a CSV string, parses the string, and constructs a ContentSearch object. Passes the ContentSearch item to the ProductAPI, which runs the actual query. Throws a ParseException if there is a problem parsing the CSV query string.
executeQueryFilename <<static>>	(String : filename) : void : throws QueryEngineException, ImportException, ParseException	Imports a CSV file and parses each line. Delegates to executeQuery() to run each query. Throws QueryEngineException, ImportException or ParseException for file handling issues or formatting problems with the CSV.

## IProductAPI <<interface>>

Interface class defining the public methods of the implementing ProductAPI. Allows customers, administrators, and application developers to search for content items. Allowed authenticated and authorized administrators to add new content to the product catalog by means of a GUID string token passed to restricted methods.

## Methods

Method Name	Signature	Description
validateAccessToken	(String : guid) : Boolean	Given the supplied access token, determine if it is valid for accessing the methods of restricted interfaces (e.g., adding content to the product catalog). <i>Note: currently this method is mocked because the implementation of the Authentication / Authorization service is delayed to a future sprint; this method will return true for any string passed.</i>
getCountryByCode	(String : code) : Country	Given a 2-character country code, search the product catalog and return any Country that matches that code.
getDeviceByID	(String : deviceId) : Device	Given a device ID string, search the product catalog and return any Device that matches that code.
importCountries	(String : guid, List<Country> countries) : void	Adds new Countries to the product catalog. <i>Restricted Interface: requires a GUID token to proceed.</i>
importDevices	(String : guid, List<Device> devices) : void	Adds new Devices to the product catalog. <i>Restricted Interface: requires a GUID token to proceed.</i>
importContent	(String : guid, List<Content> contentItems) : void	Adds new Content items to the product catalog (can be Application, Ringtone, or Wallpaper items). <i>Restricted Interface: requires a GUID token to proceed.</i>
searchContent	(ContentSearch : search) : List<Content>	Given a ContentSearch object, find all content items that match ANY of the criteria in the ContentSearch object (note this implicitly uses “OR” logic for matching conditions) and returns a list of all those matching items.
getAllApplications	() : List<Application>	Returns all the matching Application content items in the product catalog.
getAllRingtones	() : List<Ringtone>	Returns all the matching Ringtone items in the product catalog.
getAllWallpapers	() : List<Wallpaper>	Returns all the matching Wallpaper items in the product catalog.
getAllContent	() : List<Content>	Returns all the Content items in the product catalog, including all Applications, Ringtones, and Wallpapers.
getCountries	() : Set<Country>	Returns all the Countries in the product catalog.
getDevices	() : Set<Device>	Returns all the Devices in the product catalog.

getNumberContentItems	() : Integer	Returns the total number of content items in the entire product catalog.
-----------------------	--------------	--

## ProductAPI

Concrete class implementing the IProductAPI interface. Stores all the content items, countries, devices in-memory. Singleton instance.

### Methods

Method Name	Signature	Description
getInstance <<static>>	() : IProductAPI	Static getter. ProductAPI is a singleton, so the only way to get an instance of one is to call this method.

### Properties

Property Name	Type	Description
countries	Set<Country>	Holds all the Countries in the entire product catalog in-memory.
devices	Set<Device>	Holds all the Devices in the entire product catalog in-memory.
contentItems	Set<Content>	Holds all the Content items in the entire product catalog in-memory.
instance <<private>>	IProductAPI	Reference to the singleton IProductAPI instance.

### Content, Country, Manufacturer, and Device Instance Management

Due to the in-memory nature of this implementation, to optimize memory usage there should only be one instance for each unique Country, Manufacturer, Device, and Content item (where “Content” item is any concrete instance of an Application, Ringtone, or Wallpaper). This follows the Flyweight design pattern (see [http://en.wikipedia.org/wiki/Flyweight\\_pattern](http://en.wikipedia.org/wiki/Flyweight_pattern)).

## Implementation Details

The CollectionServiceAPI is implemented as a Singleton, as returned by the static getInstance() method. Following good basic design principles, the CollectionServiceAPI’s public methods are enumerated in the interface that it inherits from, ICollectionServiceAPI. Anywhere in the code that a reference to the service is expected, the interface type is returned rather than a concrete implementation.

When loading Content, Devices, and Manufacturers into the Product Catalog, it is essential to load the data in the following order:

1. Manufacturers
2. Devices

### 3. Content

Since Devices contain information on Manufacturers, the Manufacturer data must be loaded first. Likewise, since Content contains information on the Devices that the Content supports, the associated Devices must be loaded into the Product Catalog prior to the Content. Administrators may create Manufacturers, Devices, and then Content through the Product API's public "import\*" methods, each of which take a CSV file containing the appropriate data.

When Manufacturers, Devices, and Content are loaded into the Product Catalog from the "import\*" methods, new objects of those types are created and maintained in-memory in the Product Catalog (in the "contentItems" property).

To simplify searches of content, the design calls for the creation of ContentSearch objects. When a client of the Product API initiates a new content search via the searchContent() method, the client passes a single string as the search criteria. The searchCriteria() method passes control to parseSearchString(), which returns a ContentSearch object back to searchContent(). Then searchContent() performs the content search over all the contentItems in the Product Catalog based on the criteria in the ContentSearch object, and finally returns a list of all matching Content items.

### Changes from Original Design & Requirements

- Allows partial language code searches to match languages that have the criteria (for example, a content search for languages with code "fr" will match content items that have "fr\_ca", French Canadian, as their language code).
- To further emphasize the differences in content item types:
  - Wallpaper has added two properties:
    - pixelWidth
    - pixelHeight
  - Ringtone has added property:
    - durationInSeconds
- To emphasize the separation of concerns, Content items handle the validation of their own types.
- Since Content items are added to collections in the product catalog, to maintain that the ProductAPI only contains a single copy of each item (Device, Country, Application, Ringtone, Wallpaper), each of these types overrides "equals" and "hashCode" to ensure that the Set collections don't contain duplicates
- This implementation makes use of two Apache Commons JARs to facilitate more easily overriding the "equals" and "hashCode" methods in the content item classes, and also to simplify content searches (the ProductAPI's searchContent() method makes use of CollectionUtils to compare the intersection of sets).
- Content items (Application, Wallpaper, and Ringtone) and Device and Country classes override the toString() method to simplify debugging, and to more easily observe all the properties of those objects when querying

## Testing

The TestDriver class will exercise both importing collections into the Collection catalog via public methods on the CollectionServiceAPI, defining the Content items and sub-Collections that comprise them, iterating over those Collections, and searching for Collections.

A static main() method will interactively prompt the user for Countries, Devices, Content, ContentSearches, and Collections CSV files. The Collections CSV should contain definitions of collections, instructions for content to add to StaticCollections, definitions of the dynamic search content criteria on DynamicCollections, and search criteria for Collections. The main() method should call public import methods on the CollectionServiceAPI to load collections into the collection catalog and define their properties.

In addition to the sample countries.csv, devices.csv, products.csv, queries.csv, and collections.csv files that were provided, several new devices, products, content queries, and collection data were added to further test the functionality and correctness of the implementation.

## Risks

As risks are uncovered in the development process they will be documented here.

## Instructions for Compiling and Running Application

Because there are two external JAR dependencies and the package organization has been slightly reorganized from the prior sprint, the way to compile and run the application is slightly different from the original specifications.

To compile the code, run the following:

```
javac -cp ".:commons-collections4-4.0-alpha1.jar:commons-lang3-3.1.jar" cscie97/asn3/ecommerce/collection/*.java
cscie97/asn3/ecommerce/csv/*.java
cscie97/asn3/ecommerce/exception/*.java
cscie97/asn3/ecommerce/product/*.java cscie97/asn3/test/*.java
```

To run the application, run the following:

```
java -cp ".:commons-collections4-4.0-alpha1.jar:commons-lang3-3.1.jar" cscie97.asn3.test.TestDriver countries.csv devices.csv
products.csv queries.csv collections.csv
```