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 CollectionServiveAPI
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, lineIndexWhereFailed, 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 ContentSeach 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*).

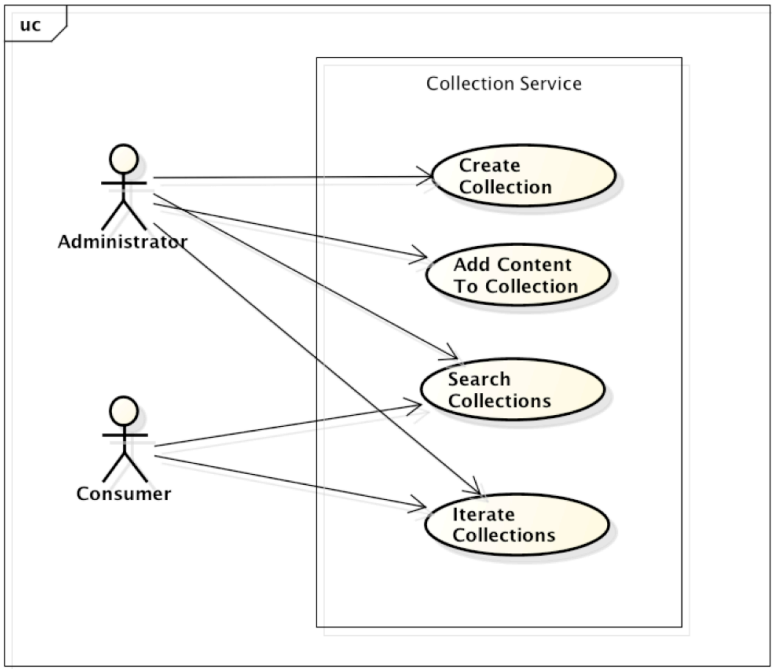
Administrators and Application Developers that have a GUID access token should be able to create new Collections (whether Static or Dynamic), and both add discrete Content (via ContentProxy objects that wrap Content items) to StaticCollections and define the search criteria for DynamicCollections. Collections 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.

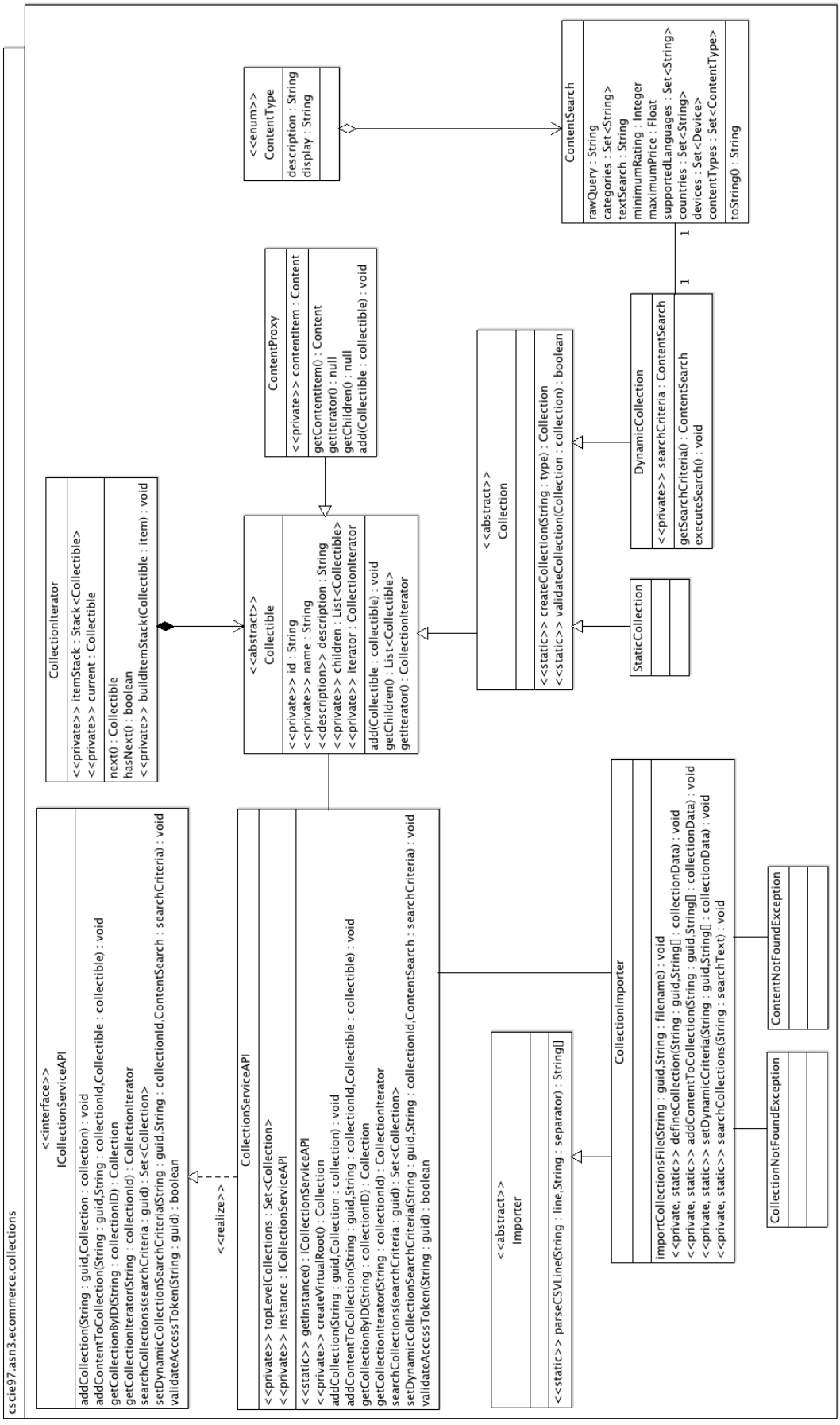
Search Collections: both Administrators and Consumers may search the Collection catalog for Collections that match the search text in either the collection's name or description, and get a set of matching Collections. In order to search for collections that match the search criteria, this use case has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

Mobile Application Store Product API Design Document
CSCI E-97 Assignment 3
David Killeffer <rayden7@gmail.com>



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". For the sake of brevity, simple accessor/mutator methods will not be documented here.

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)	Restricted interface; will validate GUID token before adding content to a collection. Adds the

	: void	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>	Finds all Collections whose Collection name or description contains any part of the searchCriteria passed. Note that the search is case-insensitive. 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.
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>	Finds all Collections whose Collection name or description contains any part of the searchCriteria passed. Note that the search is case-insensitive. 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.
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.
getInstance	() : ICollectionServiceAPI	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 that concrete classes, the return value is declared to be an interface type.
createVirtualRoot	() : Collection	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.

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 CollectionServiceAPI. 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 *actual* types may be one of: • ContentProxy (used to wrap the Content objects that get returned by the IProductAPI) • StaticCollection • DynamicCollection 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.

CollectionIterator

Allows for the iteration of Collectible items; implements Iterator. Collectibles follow a two-part Composite design pattern; at the lowest level, Collectibles can be instances of either StaticCollection or DynamicCollection, both of which inherit from Collection. These StaticCollection and DynamicCollection objects may be iterated over using this CollectionIterator. At a higher level, Collectibles may either be Collection instances or ContentProxy instances, and the two types share several common attributes.

However, since Collectible objects may have children of type Collection, they may have several children and at several layers of depth, making this iterator necessary to traverse a Collection. This iterator will traverse the Collectible depth-first.

Methods

Method Name	Signature	Description
getCurrent	() : Collectible	Convenience method to get the current item without moving the internal pointer of the iterator (calling CollectionIterator.next() will move the hidden internal pointer, but this method does not have that side-effect). Returns null if next() has not yet been called.
next	() : Collectible	Traverse the Collectible and return the next item.
buildItemStack <<private>>	(Collectible :	Called during class initialization. Populates the

	item) : void	
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.

ContentProxy

Wrapper class for Content objects that may be contained in Collections. Since all items that may be part of a Collection must be able to be easily iterated over and to reduce coupling with the ProductAPI, Content items must be wrapped so that they can inherit the same properties and be treated similarly by the CollectionIterator. However, since ContentProxy items are really just "wrapped" Content items, they should not able to add child items, get a CollectionIterator for, etc., so those inappropriate methods are overridden and return null or take no action.

Methods

Method Name	Signature	Description
getIterator <<override>>	() : CollectionIterator	Overrides method in Collectible; since ContentProxy objects may only contain single Content items, there is no need for an iterator.
getChildren <<override>>	() : List<Collectible>	Overrides method in Collectible; since ContentProxy items do not contain child collections, does nothing.
add <<override>>	(Collectible : collectible) : void	Overrides method in Collectible; since ContentProxy items do not contain child collections, does nothing.

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.

Collection <<abstract>>

Abstract class for Collections; extends Collectible. Currently supports two abstract methods: one is a factory method for the creation of either StaticCollection or DynamicCollection, and the other is for validating that the required fields for an existing Collection are valid (id, name, description).

Methods

Method Name	Signature	Description
createCollection <<static>>	(String : type) : Collection	Factory method to create an empty "shell" Collection, with no properties set. Type passed must be either “static” or “dynamic”. Only two types of Collections are currently supported: StaticCollection and DynamicCollection.
validateCollection <<static>>	(Collection : collection) : Boolean	Performs a validation check to ensure that the collection contains a minimal set of required attribute properties. Currently only requires that id, name, and description are set.
toString <<override>>	() : String	Overrides generic toString() method and print out all the properties of a Collection. Useful for debugging.

StaticCollection

Marker class for StaticCollections. StaticCollections are used to add discrete, specific Content items to a Collection. The logic for how to add content items and their properties are all contained in the parent classes Collection and its parent class, Collectible.

Methods

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

DynamicCollection

DynamicCollections contain a ContentSearch object which is executed immediately upon defining the ContentSearch. DynamicCollections may initially be created as empty "shell" objects that have not yet defined the searchCriteria property for. Once the searchCriteria

property is declared (either by calling setSearchCriteria(ContentSearch) or by passing it to the class constructor), the actual search is immediately executed by calling the IProductAPI.searchContent(ContentSearch) method and passing the search criteria object. Returned matching Content items are wrapped as ContentProxy objects because they inherit from Collectible. By executing the search criteria as soon as it is defined the DynamicCollection will have children objects available immediately after defining the search criteria, which aids in iteration (see also CollectionIterator).

Methods

Method Name	Signature	Description
setSearchCriteria	(ContentSearch : searchCriteria) : void	Defines and executes the search for Content items.
executeSearch	() : void	Executes the ContentSearch for the DynamicCollection against the IProductAPI. The returned found Content items will be wrapped as ContentProxy objects, so that all items in a Collection share the same properties and may likewise be iterated over simply, despite having different attributes and features.
toString <<override>>	() : String	Overrides generic toString() method and print out all the properties of a DynamicCollection. Useful for debugging.

Properties

Property Name	Type	Description
searchCriteria	ContentSearch	ContentSearch object used by the collection to define what criteria to use when searching for matching Content items that will first be wrapped as ContentProxy objects, and then included as children of the DynamicCollection. Required.

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).

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
```