

# Asteroid Exploration System Design Document

Date: 2013-12-20

Author: David Killeffer <rayden7@gmail.com>

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

## Introduction

This document defines the design for the Asteroid Exploration System, which is an entire software system designed to support the robotic exploration and mining of inner solar system asteroids for minerals and other resources.

More and more scarce natural resources on Earth are being irrevocably used up. As human society advances and the need for scarce resources continues to grow, human kind must increasingly look to off-planet options for meeting its needs for resources. Recent scientific studies indicate that asteroids as close as in our solar system's primary asteroid belt may provide a vast wealth of natural resources that may be extracted and brought back to Earth. The goal of the Asteroid Exploration System is to provide a rich, easy-to-use system to keep an inventory of all known and explored asteroids and their composition, as well as to remotely send spacecraft to explore and mine those asteroids that may contain precious resources and bring them back to Earth.

This document is organized into the following sections:

1. **Overview:** summarizes the problem that is being solved and why
2. **Requirements:** enumerates the various requirements for the Asteroid Exploration System as a whole, and the sub-system requirements
3. **Module Component Diagram:** illustrates the various sub-components of the system and how their interdependencies
4. **Use Cases:** enumerates high-level use cases that the Asteroid Exploration System must support
5. **Class Diagrams:** graphs of the primary classes in each sub-system and their relationships
6. **Activity Diagrams:** shows an example of the activities involved in provisioning a new Mission through the Command & Control User Interface
7. **Class Dictionaries:** a catalog of the classes that collectively comprise the Asteroid Exploration System and the various methods, properties, and attributes that define each class within each sub-system
8. **Sequence Diagrams:** shows an examples of how users interact with the Command and Control User Interface, and how that sub-system interacts with other sub-systems
9. **Command & Control User Interface Wireframes:** these wireframes show at a high level what the Command & Control UI should look like for several key screens
10. **Implementation Details:** special considerations and explanatory observations about how the classes and sub-systems relate to each other and work

11. **Changes from Original Design & Requirements:** explanations of how this implementation varies slightly from the stated requirements in `CSCIE97Assignment5AsteroidExplorationSystemRequirements.pdf`
12. **Testing:** explanation of how the classes in the Asteroid Exploration System may be tested, and explain how an example testing harness would work
13. **Risks:** identifies and enumerates any potential pitfalls or deficiencies in the current implementation and issues that might arise from this implementation

## Overview

The Asteroid Exploration System is a set of software components designed to aid mission administrators in the handling of automated robotic spacecraft that search asteroids for raw materials, water deposits, and sentient (or otherwise) living organisms. The benefits of mining asteroids for raw materials, discovery of non-Earth based water deposits, and the possibility of discovering new life are extremely tantalizing possibilities for all of humankind.

The system components of the Asteroid Exploration System are broken out into five sub-systems:

- Asteroid Inventory System
- Robotic Spacecraft Management System
- Mission Management System
- Command and Control User Interface
- Authentication Service

The Asteroid Inventory System serves as a means of cataloging all known asteroids for the purpose of archiving discoveries of minerals, water, and life. As Spacecraft make discoveries on asteroids of mineral deposits, water deposits, and organic life, the Asteroid Inventory Management system is tasked with recording and tracking these discoveries so that future mining or exploratory missions may exploit these discoveries.

The Robotic Spacecraft Management System serves as a way to provision Spacecraft for new missions to explore asteroids. Spacecraft may continually update their status to indicate fuel and communication statuses, as well as send “discovery messages” back to the Earth-based Communication Link in the Mission Management System to indicate that the Spacecraft has made a new discovery of minerals, water, or organic life.

The Mission Management System is a central control area where new missions may be defined and also started. The Mission Management System also has two important sub-components which are the Inventory Management System and the Earth-Based Communication Link. The Inventory Management System is in charge of keeping track of the overall amount of fuel available for all spacecraft not currently in the field to use, as well as the numbers of Spacecraft available (both in-field on missions as well as docked at home base), and the overall budget available to fund missions. The Earth-Based Communications Link is a software proxy that is used to receive messages sent from Spacecraft in the field; it

will take these messages and relay them to the primary Mission Management System to then take action (update Spacecraft status, update Asteroid status, etc.).

The Command and Control User Interface is a web-based front-end to the above software components. In this design a series of wireframes are presented to show the vision for how this UI should work, but the actual details are left up to the person or team implementing this design. Generally the recommendation would be to follow standard industry best-practices in the implementation of this design, using the Model-View-Controller (MVC) architectural design pattern, and probably a web framework (Ruby on Rails, PHP, Swing, ASP.NET MVC, etc. are all potential options, but there are many more). The Command and Control UI makes use of all the other software components in the Model and Controller layers of an MVC architecture.

The final primary component is the re-use of the preexisting Authentication Service API. This service allows users to be defined that have administrative rights to carry out the method calls exposed via each of the primary software system's public APIs. Currently the functional requirements only necessitate a single level of user authentication; either a user is authorized to use all of the restricted interface methods, or authorized for none. The current version of the Authentication Service API will actually allow much finer-grained access control (allows for Service definitions, Permissions to individual methods, and Roles which collect several Permissions into a defined package of Permissions). The service is used to authenticate and authorize users for all public methods - this can be seen by all restricted interface methods requiring a "guid" token be passed prior to being able to use the method, at which time the Authentication Service API is checked to ensure the user has rights.

## Asteroid Exploration System Requirements

This section defines the requirements for the overall Asteroid Exploration System and its sub-components. The requirements include:

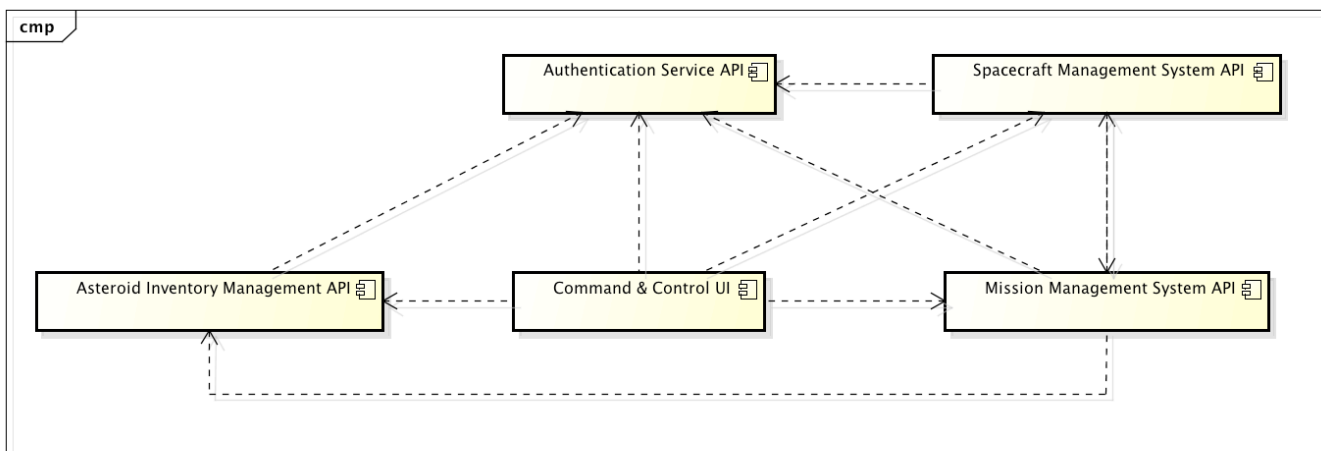
- Components should be loosely coupled and internally cohesive (this requirement is illustrated though the Module Component Diagram)
- Persistent data should be saved to a permanent data store (database); this requirement is satisfied though the class elements that should be persisted by implementing a "persist()" method that writes the serializable object properties to a datastore
- Asteroid Inventory Management System requirements:
  - Asteroids must support a unique identifier, which should also serve as the primary key of the object in the persistent database (the "persist()" method on the Asteroid class uses this key)
  - Other specific fields that Asteroids must support are enumerated in the CSCIE97Assignment5AsteroidExplorationSystemRequirements.pdf document (omitted here for brevity)
  - Include a method for listing all known Asteroids
  - Include a method for obtaining a specific Asteroid by its ID

- Include a method to Create New Asteroids
- Include a method to Update Existing Asteroids
- Include a method to Query for Asteroids based on their properties
- All public methods of the Asteroid component have restricted access (e.g., must supply a valid GUID access token from the Authentication Service API)
- Robotic Spacecraft Management System requirements:
  - Spacecraft must support a unique identifier, which should also serve as the primary key of the object in the persistent database (the “persist()” method on the Spacecraft class uses this key)
  - Other specific fields that Spacecraft must support are enumerated in the CSCIE97Assignment5AsteroidExplorationSystemRequirements.pdf document (omitted here for brevity)
  - Include a method for listing all Spacecraft
  - Include a method for obtaining a specific Spacecraft by its ID
  - Include a method to Create New Spacecraft
  - Include a method to Update Existing Spacecraft
  - All public methods of the Spacecraft component have restricted access (e.g., must supply a valid GUID access token from the Authentication Service API)
  - Spacecraft must also be able to send messages to the Mission Management System when they made a new discovery of minerals, water, or organic life
  - All public methods of the Robotic Spacecraft Management component have restricted access (e.g., must supply a valid GUID access token from the Authentication Service API)
- Mission Management System requirements:
  - The Mission Management System is responsible for creating new Missions and managing existing ones
  - Missions must support a unique identifier, which should also serve as the primary key of the object in the persistent database (the “persist()” method on the Mission class uses this key)
  - Other specific fields that Missions must support are enumerated in the CSCIE97Assignment5AsteroidExplorationSystemRequirements.pdf document (omitted here for brevity)
  - Must also support an Inventory Manager, which bears responsibility for tracking the available fuel resources, number of deployed/inactive Spacecraft, and total available budget for all Missions
  - As new Missions are created and launched, must inform the Inventory Manager and decrement available resources of fuel, budget money, and available Spacecraft
  - Missions must also maintain contact with Spacecraft via an Earth-Bound Communications Link; Spacecraft may send messages to the Communications Link informing it of new discoveries made on the Asteroids they are exploring, and then the Communications Link informs the Mission Management system, which should update the Asteroid (add new discovery information to it and save it) and Mission status as appropriate

- All public methods of the Mission Management System component have restricted access (e.g., must supply a valid GUID access token from the Authentication Service API)
- Command and Control User Interface requirements:
  - Must provide a graphical user interface (GUI) for interacting with the Mission Management System, Asteroid Inventory System, Robotic Spacecraft Management System, and Authentication System
  - Users that have logged in use their credentials for interactions with the other system components for Authentication/Authorization purposes
  - Must support the following use cases:
    - Login / Logout
    - Create New Mission
    - Monitor and Update Mission Status
    - Monitor and Update Asteroid Status
    - Monitor and Update Spacecraft Status
    - Monitoring Inventory Management resources (fuel, overall budget, available/deployed Spacecraft totals)
    - Monitoring the Status of Earth-Based Communication Link
    - Monitoring incoming messages from Spacecraft
- Authentication System requirements:
  - The requirements for the Authentication System remain the same as they did for the Mobile Application Store project, since this component is simply being re-used here without any changes

## Module Component Diagram

This diagram shows how the individual sub-system components within the application interact with each other.



powered by Astah

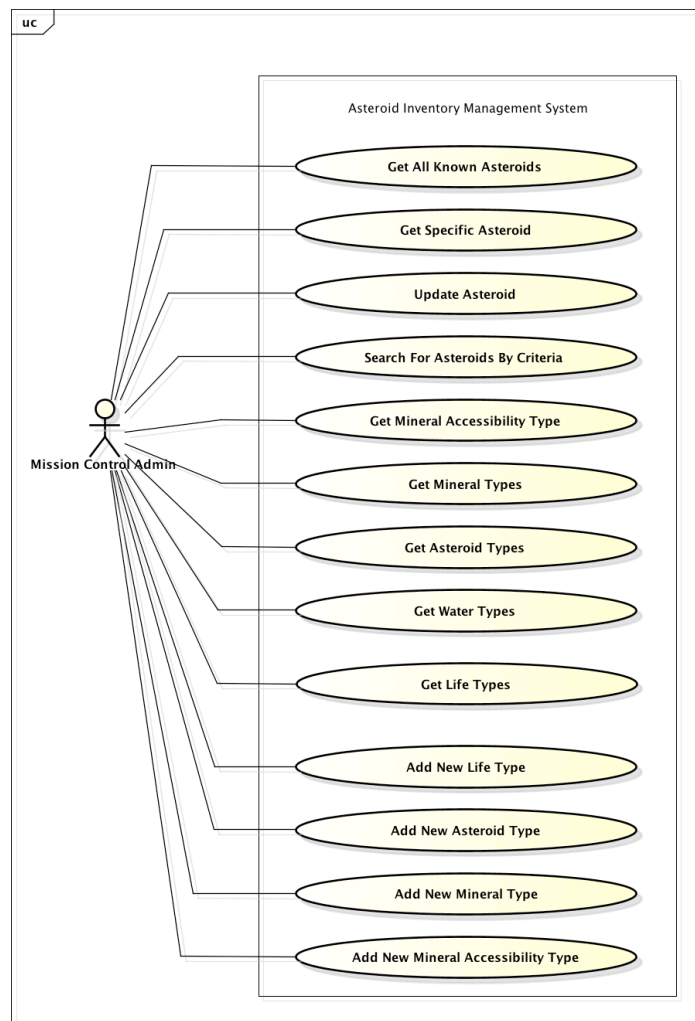
As shown, the Command & Control UI has dependencies on all other modules; this should not be surprising as it represents the user-facing front end to the entire application. The Asteroid Inventory Management system stands as the most isolated component, and can exist

largely on it's own, except for requiring authentication/authorization checks. The Spacecraft Management System API also requires checking the Authentication System for restricted interface method access, but also on the Mission Management System for communicating resource discovery messages and also Spacecraft status messages to the Mission Management System. The Mission Management System has dependencies on both the Spacecraft Management System and the Asteroid Inventory Management systems, because it needs to modify properties on the Spacecraft when missions start/end and their status changes over the course of a mission, and also it needs to update the discoveries found on the Asteroid the Spacecraft is exploring.

## Use Cases

The following use case diagram shows the major functional use cases that each type of user of the Authentication Service can perform. Because there are many use cases, they will be displayed in separate diagrams for each sub-system.

### Asteroid Inventory Management System



powered by Astah

**Get All Known Asteroids:** list all the Asteroid objects in the system. Requires a GUID authentication token from the Authentication System.

**Get Specific Asteroid:** support obtaining a specific Asteroid object by it's ID. Requires a GUID authentication token from the Authentication System.

**Update Asteroid:** overwrite the properties of an Asteroid with new values, and write the updates to the persistent datastore. Requires a GUID authentication token from the Authentication System.

**Search For Asteroids By Criteria:** allow searching for all matching Asteroids that match a given set of search criteria. Requires a GUID authentication token from the Authentication System.

**Get Mineral Accessibility Types:** return a list of all the enumeration objects for AccessibilityType

**Get Mineral Types:** return a list of all the enumeration objects for MineralType

**Get Asteroid Types:** return a list of all the enumeration objects for AsteroidType

**Get Water Types:** return a list of all the enumeration objects for WaterType

**Get Life Types:** return a list of all the enumeration objects for LifeType

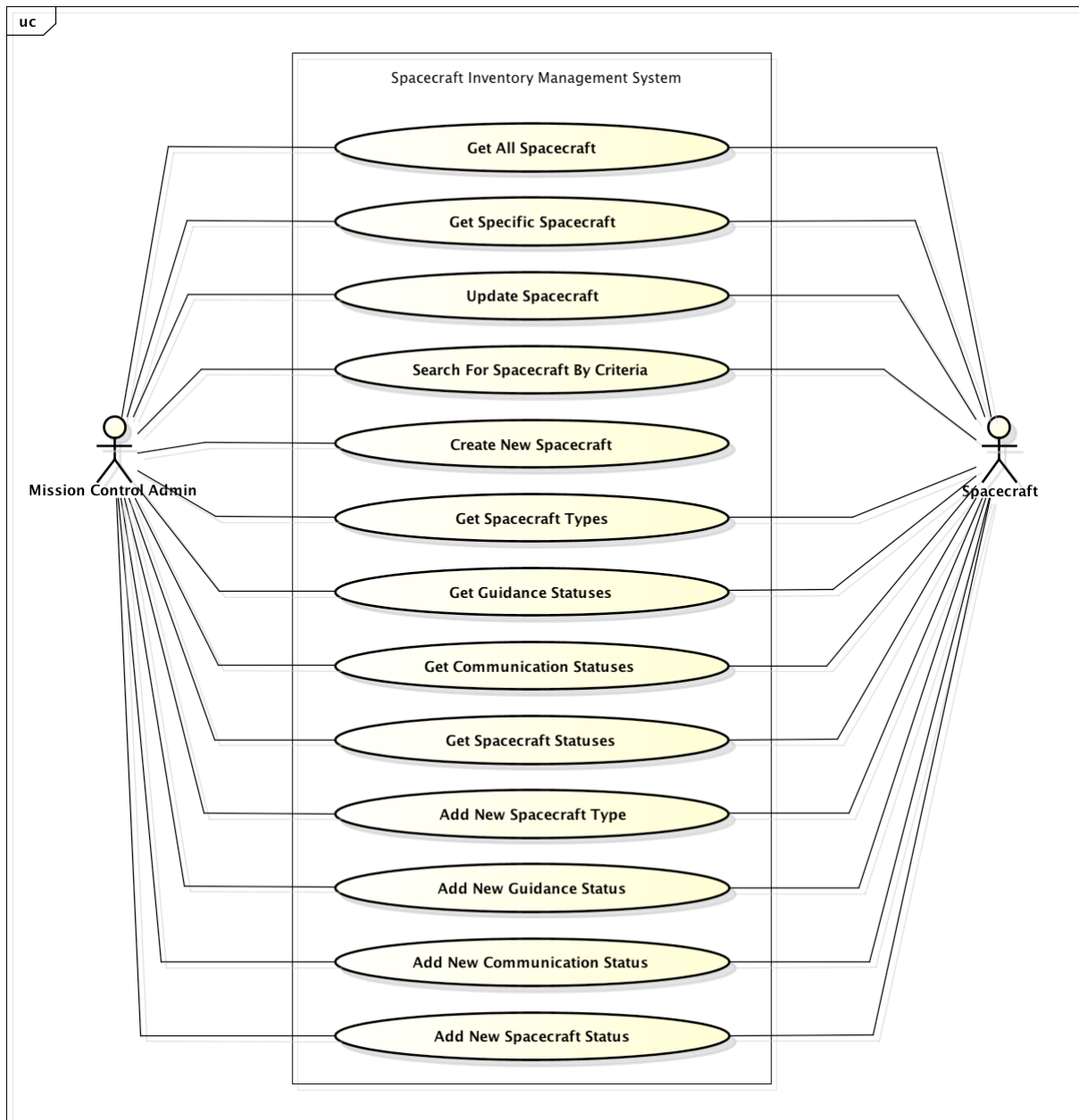
**Add New Life Type:** allows adding a new type to the LifeType enumeration that is added to the persistent datastore

**Add New Asteroid Type:** allows adding a new type to the AsteroidType enumeration that is added to the persistent datastore

**Add New Mineral Type:** allows adding a new type to the MineralType enumeration that is added to the persistent datastore

**Add New Mineral Accessibility Type:** allows adding a new type to the AccessibilityType enumeration that is added to the persistent datastore

## Robotic Spacecraft Management System



powered by Astah

**Get All Spacecraft:** list all the Spacecraft objects in the system. Requires a GUID authentication token from the Authentication System.

**Get Specific Spacecraft:** support obtaining a specific Spacecraft object by it's ID. Requires a GUID authentication token from the Authentication System.

**Update Spacecraft:** overwrite the properties of a Spacecraft with new values, and write the updates to the persistent datastore. Requires a GUID authentication token from the Authentication System.



**Search For Spacecraft By Criteria:** allow searching for all matching Spacecraft that match a given set of search criteria. Requires a GUID authentication token from the Authentication System.

**Create New Spacecraft:** allow the creation and saving to the persistent datastore of new Spacecraft objects. Requires a GUID authentication token from the Authentication System.

**Get Spacecraft Types:** return a list of all the enumeration objects for SpacecraftType

**Get Guidance Statuses:** return a list of all the enumeration objects for GuidanceStatus

**Get Communication Statuses:** return a list of all the enumeration objects for CommunicationStatus

**Get Spacecraft Statuses:** return a list of all the enumeration objects for SpacecraftStatus

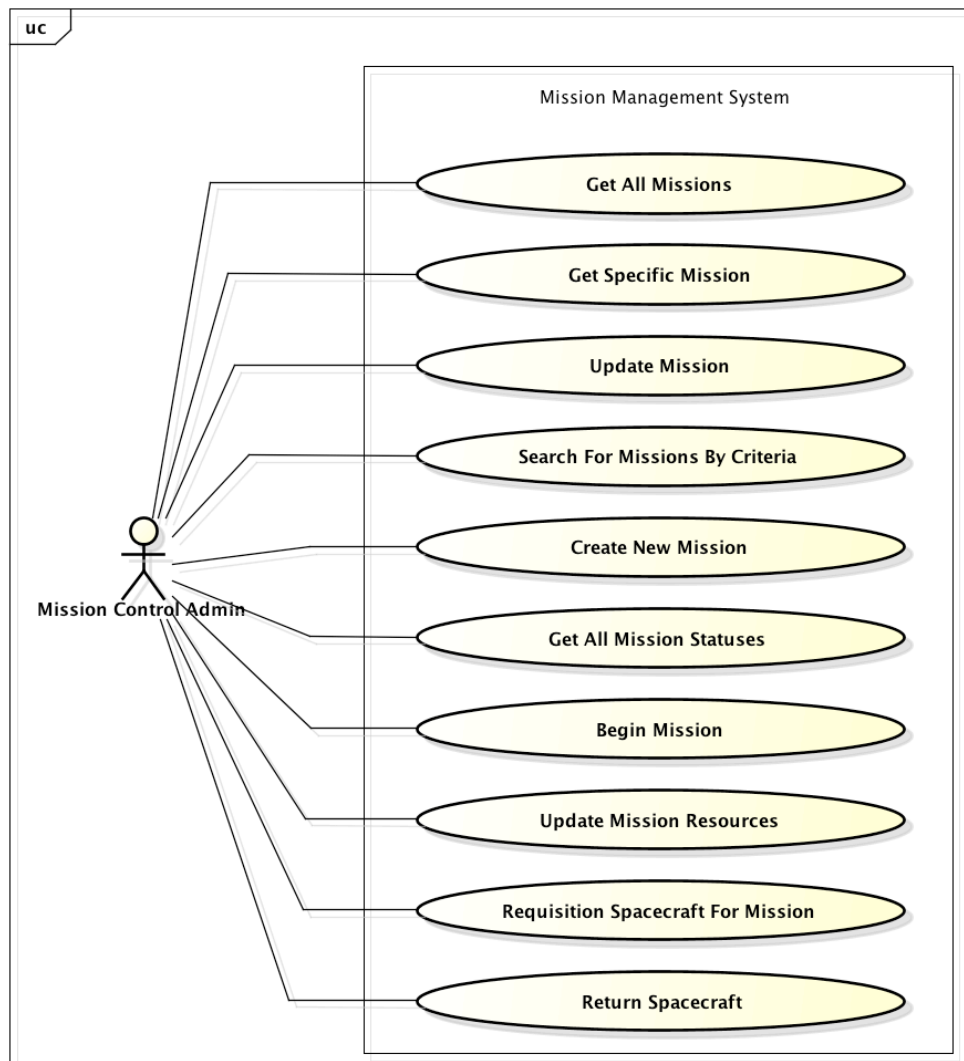
**Add New Spacecraft Type:** allows adding a new type to the SpacecraftType enumeration that is added to the persistent datastore

**Add New Guidance Status:** allows adding a new type to the GuidanceStatus enumeration that is added to the persistent datastore

**Add New Communication Status:** allows adding a new type to the CommunicationStatus enumeration that is added to the persistent datastore

**Add New Spacecraft Status:** allows adding a new type to the SpacecraftStatus enumeration that is added to the persistent datastore

## Mission Management System



powered by Astah

**Get All Missions:** list all the Mission objects in the system. Requires a GUID authentication token from the Authentication System.

**Get Specific Mission:** support obtaining a specific Mission object by it's ID. Requires a GUID authentication token from the Authentication System.

**Update Mission:** overwrite the properties of a Mission with new values, and write the updates to the persistent datastore. Requires a GUID authentication token from the Authentication System.

**Search For Missions By Criteria:** allow searching for all matching Missions that match a given set of search criteria. Requires a GUID authentication token from the Authentication System.

**Create New Mission:** allow the creation and saving to the persistent datastore of new Mission objects. Requires a GUID authentication token from the Authentication System.

**Get All Mission Statuses:** return a list of all the enumeration objects for MissionStatus

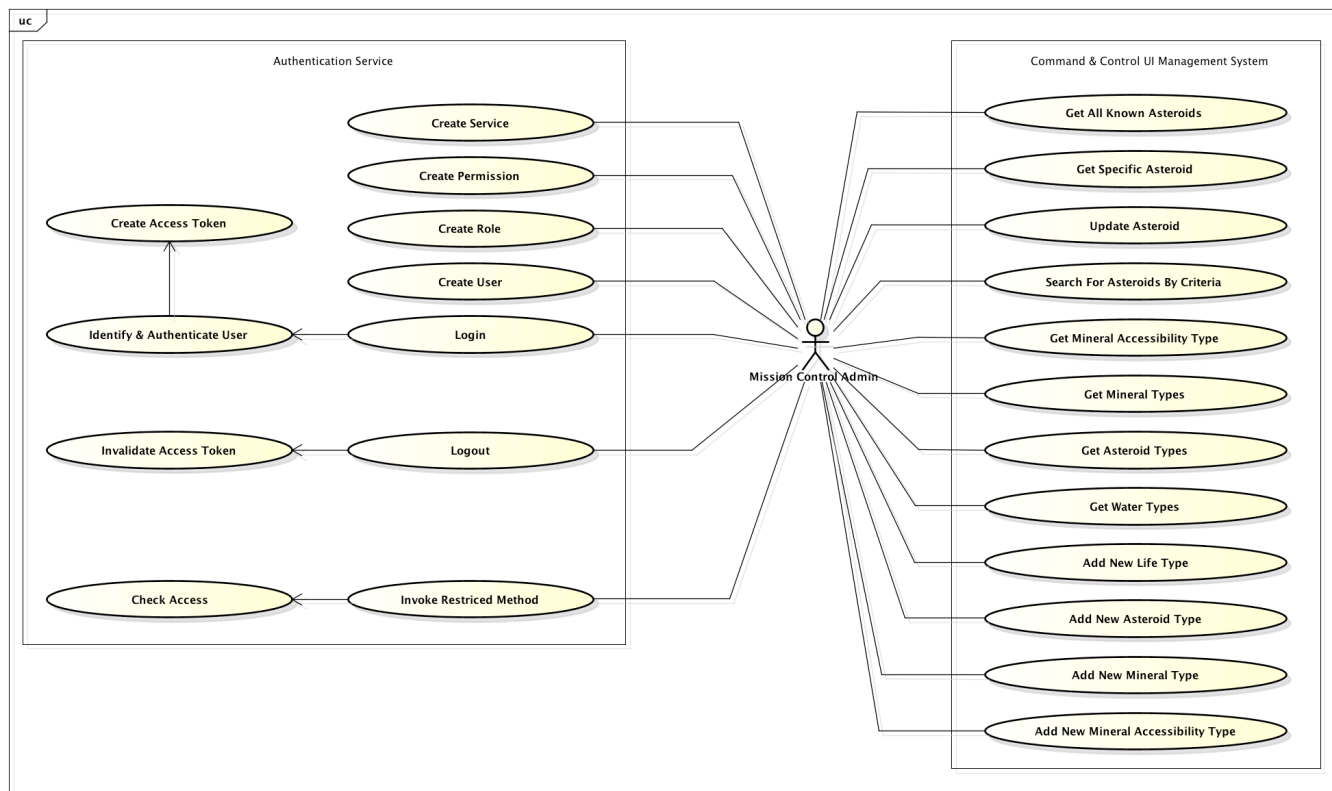
**Begin Mission:** officially launch a Mission and all the Spacecraft connected to it; has side effect of altering resources available in Inventory Manager

**Update Mission Resources:** update Spacecraft and Asteroid statuses and resources based on an incoming Message from the Spacecraft

**Requisition Spacecraft For Mission:** makes selected Spacecraft for Mission unavailable to be selected for other Missions

**Return Spacecraft:** returns Spacecraft back into the available Inventory of Spacecraft that may be selected for other/future Missions

## Command & Control User Interface



powered by Astah

Most of these use cases will be illustrated through the wireframes of the Command & Control UI later in the document. The Create Service, Create Permission, Create Role, and Create User methods of the Authentication Service are accessible to the Mission Control Administrator through the code, but the current design requirements for the Command & Control UI do not declare that they need to be supported through the UI. Mission Control

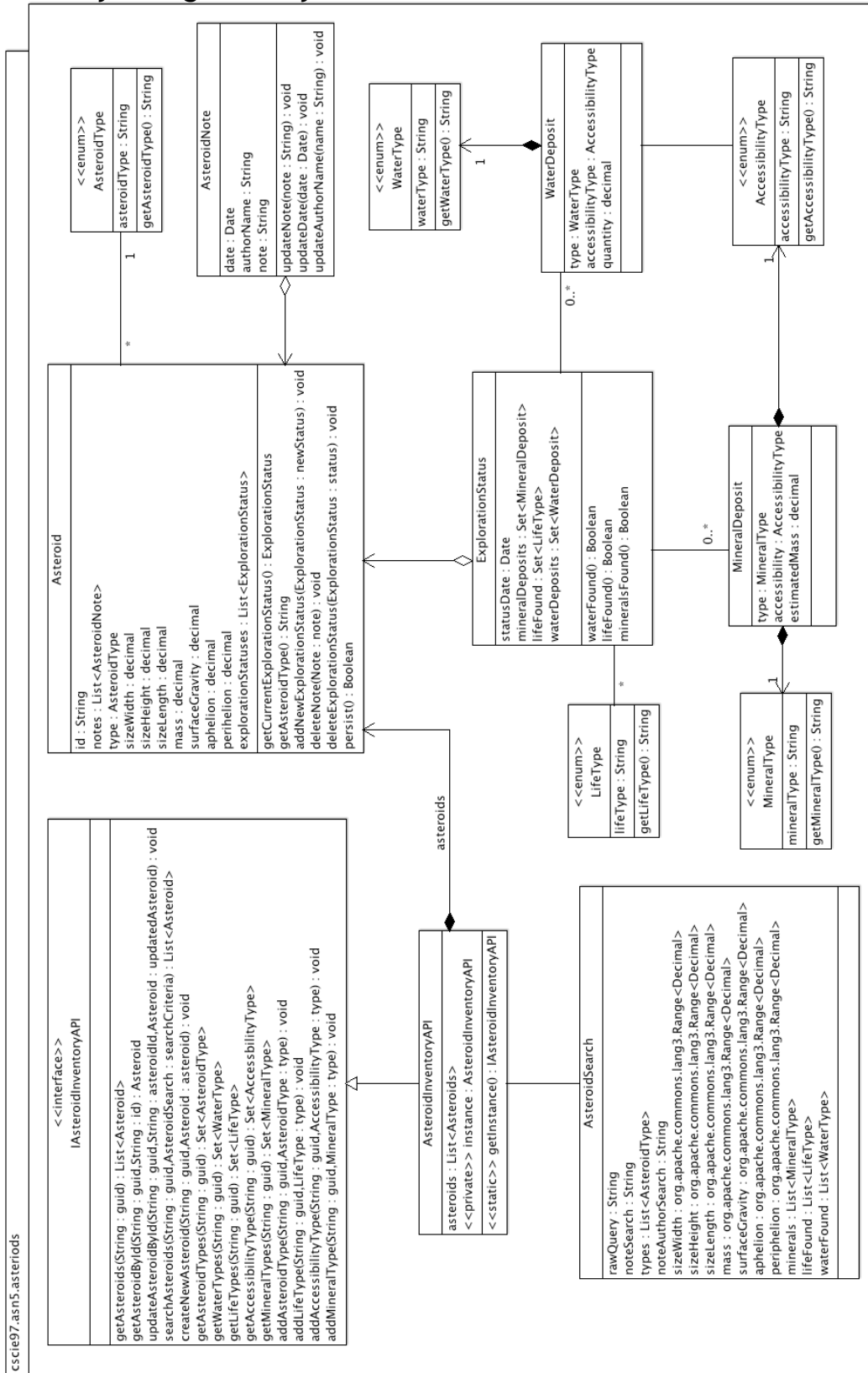
Administrators must first be “logged in” prior to being able to call any of the methods that are exposed through the Command & Control UI except for “Login” (which itself authenticates against the Authorization System).

## **Class Diagrams**

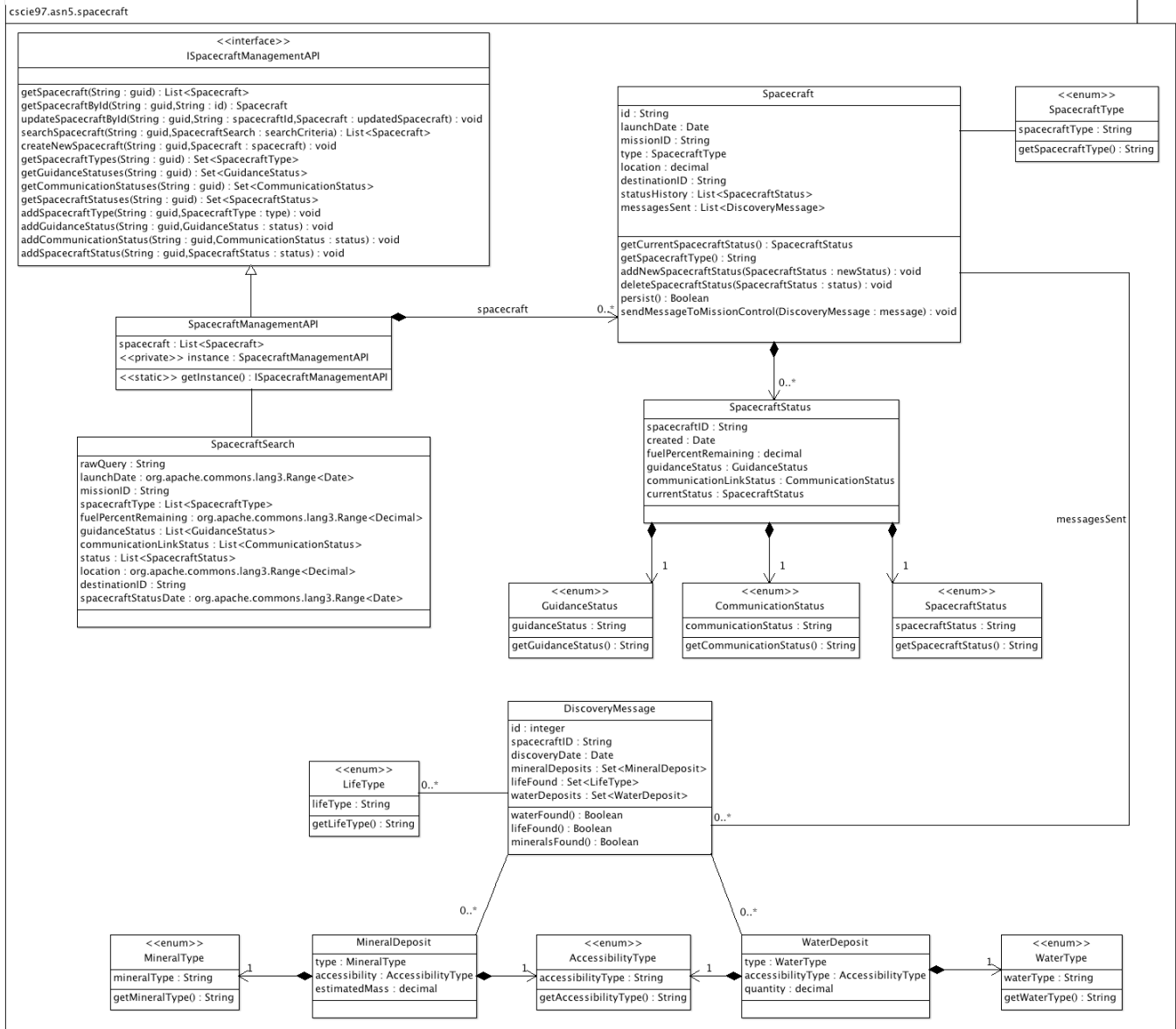
Each sub-system will have it’s own class diagram. Note that there is not a class diagram for the Command & Control UI; the expectation is that this system will use best practices during development and use a Model-View-Controller (MVC) design pattern and supporting web framework. The selection of what supporting web framework/library is left to the discretion of the implementing team.

Exposition of each class, interface, and enumeration in each sub-system’s class diagram will be explained in the Class Dictionary in the next section.

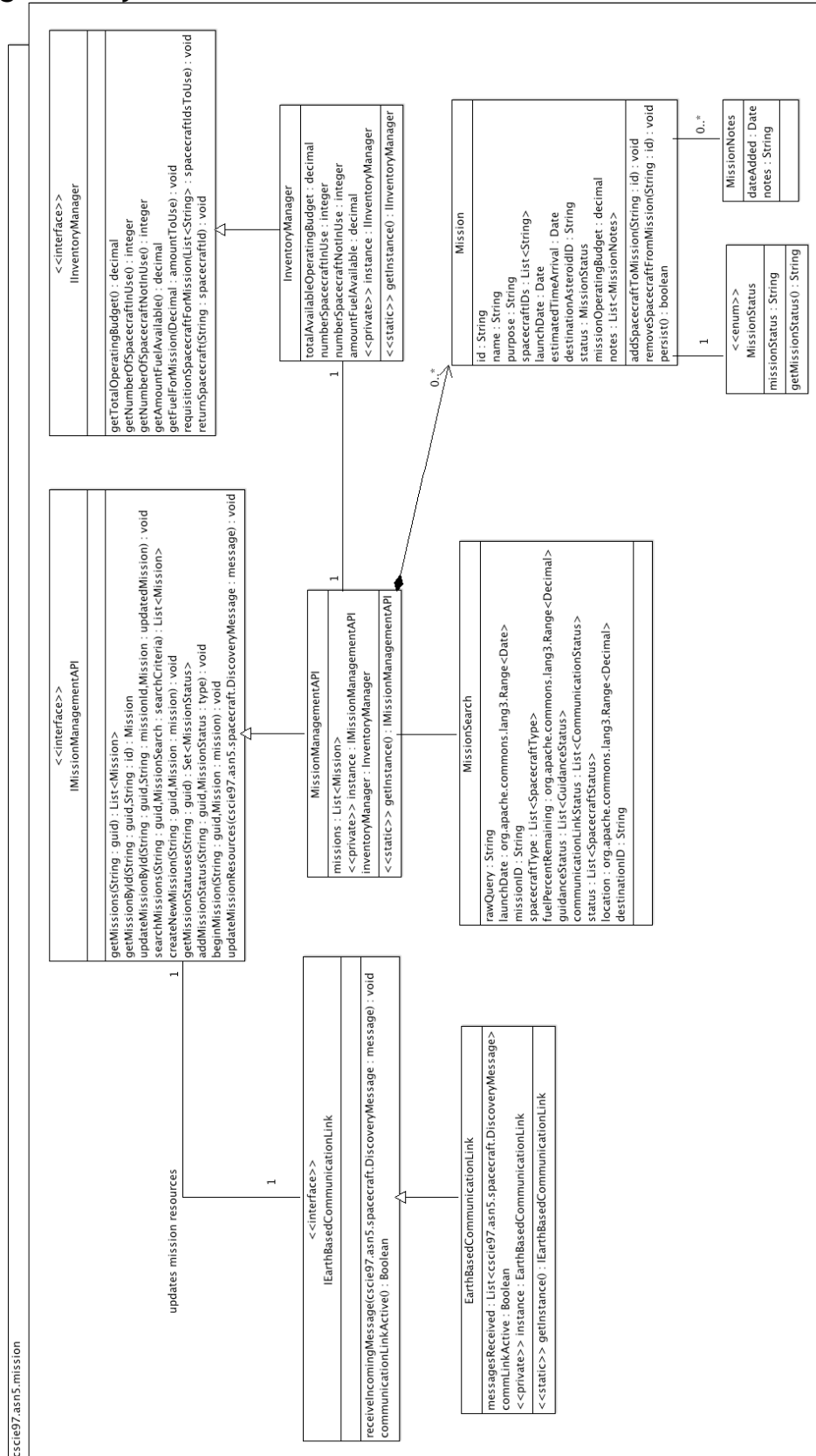
## Asteroid Inventory Management System



## Robotic Spacecraft Management System

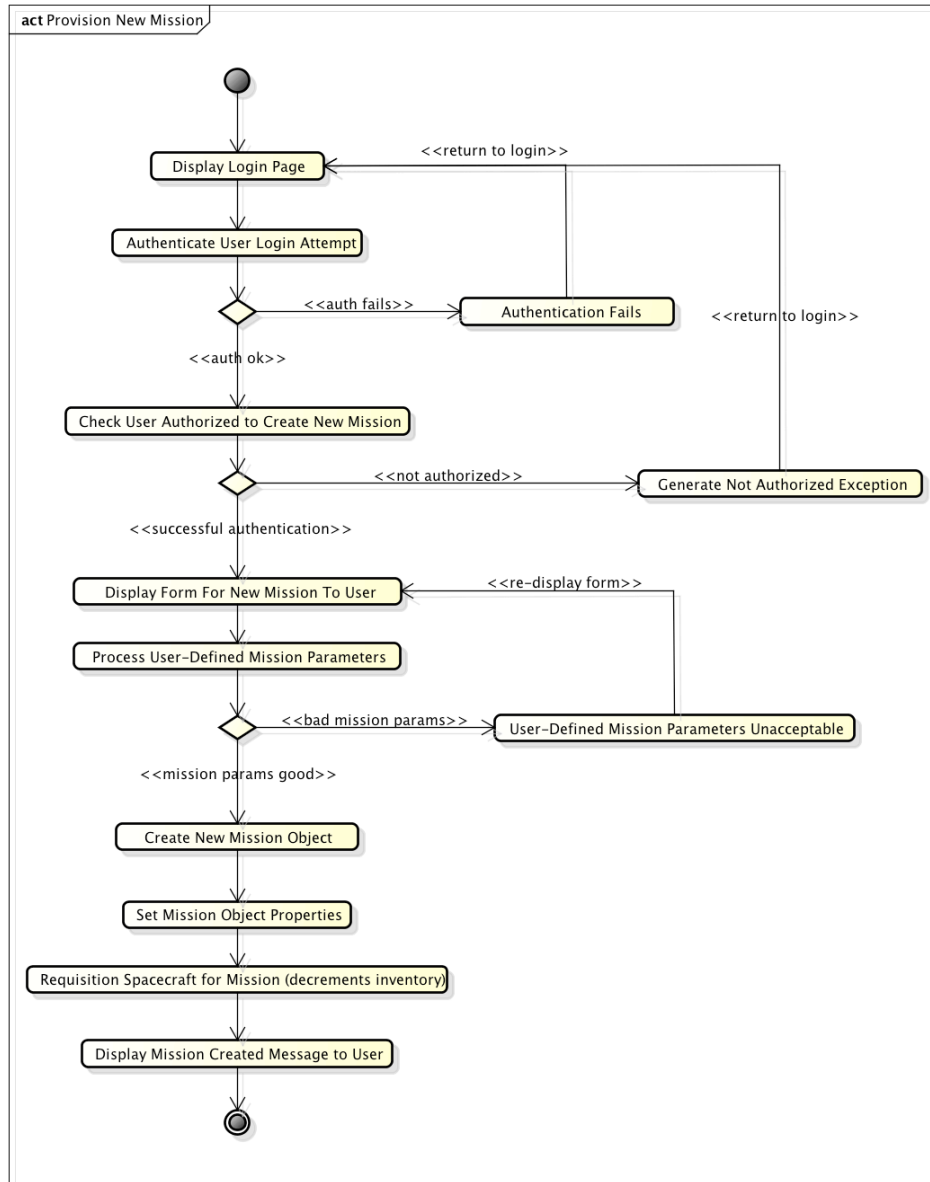


## Mission Management System



## Activity Diagram

The following example activity diagram shows the logical process to provision a new Mission by users of the system.



powered by Astah

## Class Dictionaries

This section specifies the class dictionaries for each sub-system in the Asteroid Exploration System. For the sake of brevity, simple accessor/mutator methods will not be documented here, nor will overridden methods in classes implementing interfaces or abstract classes.



## Package: cscie97.asn5.asteroids

### ***AsteroidInventoryAPI*** <<interface>>

This interface defines the primary service contract for any classes that intend to implement these methods and act as the concrete implementation of the AsteroidInventoryAPI. All methods require authenticated access via the Authorization Service API; methods must pass a String GUID token, which is checked before continuing. Allows authenticated users access to manage all aspects of Asteroids, including creation, listing asteroids, searching, and updating Asteroids. This interface and it's implementing class, AsteroidInventoryAPI, collectively work as a façade design pattern - only the public methods here are accessible outside the package.

#### ***Methods***

Method Name	Signature	Description
getAsteroids	(String : guid) : List<Asteroid>	Returns a list of all the Asteroids under management by the API.
getAsteroidById	(String : guid, String : id) : Asteroid	Retrieves the corresponding Asteroid object from the inventory
updateAsteroidById	(String : guid, String : asteroidID, Asteroid : updatedAsteroid) : void	Finds the Asteroid in the inventory with the specified ID, and overwrites that object's properties with those of the updatedAsteroid. Should also persist the changes to the database.
searchAsteroids	(String : guid, AsteroidSearch : searchCriteria) : List<Asteroid>	Given the supplied AsteroidSearch criteria object, find and return all Asteroids that match any of the supplied criteria in the search object.
createNewAsteroid	(String : guid, Asteroid : asteroid) : void	Clients may create new Asteroid objects independently, but this method must be called to add the Asteroid to the inventory (also persists the newly created Asteroid to the database).
getAsteroidTypes	(String : guid) : Set<AsteroidType>	Returns all the enumeration values of AsteroidType.
getWaterTypes	(String : guid) : Set<WaterType>	Returns all the enumeration values of WaterType.
getLifeTypes	(String : guid) : Set<LifeType>	Returns all the enumeration values of LifeType.
getAccessibilityTypes	(String : guid) : Set<AccessibilityType>	Returns all the enumeration values of AccessibilityType.
getMineralTypes	(String : guid) :	Returns all the enumeration values of

	Set<MineralType>	MineralType.
addAsteroidType	(String : guid, AsteroidType : type) : void	Adds the new AsteroidType to the database; subsequent application loads will pick up this new available AsteroidType.
addLifeType	(String : guid, LifeType : type) : void	Adds the new LifeType to the database; subsequent application loads will pick up this new available LifeType.
addAccessibilityType	(String : guid, AccessibilityType : type) : void	Adds the new AccessibilityType to the database; subsequent application loads will pick up this new available AccessibilityType.
addMineralType	(String : guid, MineralType : type) : void	Adds the new MineralType to the database; subsequent application loads will pick up this new available MineralType.

### ***AsteroidInventoryAPI, implements IAsteroidInventoryAPI***

This concrete implementation of the IAsteroidInventoryAPI follows the Singleton design pattern; to obtain an instance to use, the getInstance() method will return the sole instance of the object.

#### ***Properties***

Property Name	Type	Description
asteroids	List<Asteroid>	Contains all the Asteroids in the inventory. Asteroids are initially loaded from the database behind-the-scenes when the application loads and the instance is initialized.
instance <<private>>	AsteroidInventoryAPI	Singleton instance of the API; only one exists at a time

#### ***Methods***

Method Name	Signature	Description
getInstance <<static, synchronized>>	() : IAsteroidInventoryAPI	Used to obtain a Singleton instance of the API for client use.

### ***AsteroidSearch***

Allows users to create a set of search criteria that richly encapsulates all the properties of an Asteroid. Since Asteroids have several numeric and date based properties, utilizes Apache Commons *org.apache.commons.lang3.Range<T>* class to create range-based search criteria (for example, this would allow searching for all Asteroids with a mass of between 120 and 999 tons, etc.). The search criteria properties are explicitly “OR”, to be as inclusive as possible.

### *Properties*

Property Name	Type	Description
rawQuery	String	For text-based searches, simply contains the entire text that the user wishes to search for in String fields on the Asteroid. May optionally be logged (or saved in the database) to capture a history of searches executed.
noteSearch	String	What to search for in the “notes” fields of Asteroids
types	List<AsteroidType>	Find Asteroids that match at least 1 of these AsteroidTypes
noteAuthorSearch	String	Find Asteroids where the author name matches this property
sizeWidth	org.apache.commons.lang3.Range<Decimal>	Find all Asteroids that have a sizeWidth property in between the range supplied (inclusive)
sizeHeight	org.apache.commons.lang3.Range<Decimal>	Find all Asteroids that have a sizeHeight property in between the range supplied (inclusive)
sizeLength	org.apache.commons.lang3.Range<Decimal>	Find all Asteroids that have a sizeLength property in between the range supplied (inclusive)
mass	org.apache.commons.lang3.Range<Decimal>	Find all Asteroids that have a mass property in between the range supplied (inclusive)
surfaceGravity	org.apache.commons.lang3.Range<Decimal>	Find all Asteroids that have a surfaceGravity property in between the range supplied (inclusive)
aphelion	org.apache.commons.lang3.Range<Decimal>	Find all Asteroids that have a aphelion property in between the range supplied (inclusive)
periphelion	org.apache.commons.lang3.Range<Decimal>	Find all Asteroids that have a periphelion property in between the range supplied (inclusive)
minerals	List<MineralType>	Find all Asteroids that have at least 1 of these MineralTypes

		found
lifeFound	List<LifeType>	Find all Asteroids that have at least 1 of these LifeTypes found
waterFound	List<WaterType>	Find all Asteroids that have at least 1 of these WaterTypes found

## ***Asteroid***

Asteroids model the items that Spacecraft are exploring. Each Asteroid contains a history of notes that are added by Mission Control Admin staff; this will support a nice historical view of what staff have observed about the Asteroid over time. Asteroids may also have a list of ExplorationStatuses that are updated as Spacecraft make new discoveries and transmit them to the Mission Management System, which then updates these ExplorationStatuses. This too supports a rich historical view of the Asteroid, denotes what was found, and when (since discoveries take place at different times, in different quantities, etc.). Asteroids may be persisted to the database; they have a method here called “persist()” which should serialize the Asteroid and write the pertinent information to the database. All properties of the Asteroid should persist, and the database key should be the ID field.

## ***Properties***

Property Name	Type	Description
Id	String	Unique identifier for the Asteroid, and also the primary key in the database.
notes	List<AsteroidNote>	A list of notes that mission control staff write about the Asteroid. AsteroidNotes are written to a separate database table but linked to Asteroid objects by the ID.
type	AsteroidType	The particular type that this Asteroid is. Each Asteroid can only be of a single type.
sizeWidth	Decimal	The estimated width of the Asteroid in meters.
sizeHeight	Decimal	The estimated height of the Asteroid in meters.
sizeLength	Decimal	The estimated length of the Asteroid in meters.
mass	Decimal	The estimated mass of the Asteroid in tons.
surfaceGravity	Decimal	The estimated number of Earth gravities that are exerted over objects on the surface of the

		Asteroid.
Aphelion	Decimal	The estimated distance from the Sun in AUs
Perihelion	Decimal	The estimated distance from the Sun in AUs
explorationStatuses	List<ExplorationStatus>	History of all prior exploration discoveries made on the asteroid, including minerals found, water, and organic life found.

### **Methods**

Method Name	Signature	Description
getCurrentExplorationStatus	() : ExplorationStatus	Returns the most recent ExplorationStatus based on the statusDate.
getAsteroidType	() : String	Returns the String type of the AsteroidType the Asteroid is
addNewExplorationStatus	(ExplorationStatus : newStatus) : void	Adds a new ExplorationStatus to the list. When added the new status should also be persisted to the database.
deleteNote	(AsteroidNote : note) : void	Deletes the supplied AsteroidNote from the object and from the database.
deleteExplorationStatus	(ExplorationStatus : status) : void	Deletes the supplied ExplorationStatus from the object and from the database.
persist	() : Boolean	Save the Asteroid and all associated objects (AsteroidNotes, ExplorationStatuses) to the database; returns true if successful, false otherwise

### **AsteroidNote**

Contains the observation notes of a Mission Control Administrator on the Asteroid. AsteroidNotes are saved to the database in their own table, and have a many-to-one relationship with Asteroid (one Asteroid, many AsteroidNotes). Any time the parent Asteroid is saved to the database, the AsteroidNotes are also updated.

### **Properties**

Property Name	Type	Description
date	Date	When the note was last updated
authorName	String	Name of the note author
note	String	The body of the author's observational note on the Asteroid.

### **Methods**

Method Name	Signature	Description
updateNote	(String : note) : void	Updates the note with the new text
updateDate	(Date : date) : void	Modifies the date of the note
updateAuthorName	(String : name) : void	Modifies the name of the Author

### ***ExplorationStatus***

Represents a new discovery of either water, organic life, or minerals on the Asteroid. ExplorationStatus objects are stored in their own database table and each one may correspond to a single Asteroid (Asteroids can have many associated ExplorationStatuses). Since Asteroids may have many of these ExplorationStatuses, they may develop a rich history of when things were discovered on the Asteroid, in what quantities, etc.

#### ***Properties***

Property Name	Type	Description
statusDate	Date	When the initial discovery was made
mineralDeposits	Set<MineralDeposit>	A collection of all the MineralDeposit objects that were found
lifeFound	Set<LifeType>	A collection of all the LifeTypes that were found
waterDeposits	Set<WaterDeposit>	A collection of all the WaterDeposits that were found

#### ***Methods***

Method Name	Signature	Description
waterFound	() : Boolean	Convenience method to determine if there was any water found in this ExplorationStatus
lifeFound	() : Boolean	Convenience method to determine if there was any life found in this ExplorationStatus
mineralsFound	() : Boolean	Convenience method to determine if there was any minerals found in this ExplorationStatus

### ***MineralDeposit***

Represents a new mineral deposit found on the Asteroid. Each MineralDeposit is saved to it's own database table.

#### ***Properties***

Property Name	Type	Description
type	MineralType	The MineralType enum value that this discovery

		corresponds to
Accessibility	AccessibilityType	How easy or hard it will be to mine the Asteroid for the mineral type found
estimatedMass	Decimal	Estimated amount of the mineral found in tons

### ***WaterDeposit***

Represents a new water deposit found on the Asteroid. Each WaterDeposit is saved to it's own database table.

#### ***Properties***

Property Name	Type	Description
type	WaterType	The WaterType enum value that this discovery corresponds to
accessibility	AccessibilityType	How easy or hard it will be to obtain the water from the Asteroid (e.g., is the water underground, on top of a mountain, etc.)
quantity	Decimal	Estimated amount of the water found in the unit applicable to it's type (e.g., for liquid water this would be gallons, for ice it might be in tons, etc.)

### ***AsteroidType <<enum>>***

An enumeration representing the types of Asteroids that are known. Each enumeration will correspond to it's own row in a distinct database table.

#### ***Methods***

Method Name	Signature	Description
getAsteroidType	() : String	Returns the String name of the enumeration property

#### ***Properties***

Property Name	Type	Description
A_TYPE	String	Used by Asteroid; asteroidType is "A-Type" (see <a href="http://en.wikipedia.org/wiki/A-type_asteroid">http://en.wikipedia.org/wiki/A-type_asteroid</a> )
C_TYPE	String	Used by Asteroid; asteroidType is "C-Type" (see <a href="http://en.wikipedia.org/wiki/C-type_asteroid">http://en.wikipedia.org/wiki/C-type_asteroid</a> )
K_TYPE	String	Used by Asteroid; asteroidType is "K-Type" (see <a href="http://en.wikipedia.org/wiki/K-type_asteroid">http://en.wikipedia.org/wiki/K-type_asteroid</a> )
L_TYPE	String	Used by Asteroid; asteroidType is "L-Type" (see

		<a href="http://en.wikipedia.org/wiki/L-type_asteroid">http://en.wikipedia.org/wiki/L-type_asteroid</a> )
M_TYPE	String	Used by Asteroid; asteroidType is “M-Type” (see <a href="http://en.wikipedia.org/wiki/M-type_asteroid">http://en.wikipedia.org/wiki/M-type_asteroid</a> )
Q_TYPE	String	Used by Asteroid; asteroidType is “Q-Type” (see <a href="http://en.wikipedia.org/wiki/Q-type_asteroid">http://en.wikipedia.org/wiki/Q-type_asteroid</a> )
R_TYPE	String	Used by Asteroid; asteroidType is “R-Type” (see <a href="http://en.wikipedia.org/wiki/R-type_asteroid">http://en.wikipedia.org/wiki/R-type_asteroid</a> )
S_TYPE	String	Used by Asteroid; asteroidType is “S-Type” (see <a href="http://en.wikipedia.org/wiki/S-type_asteroid">http://en.wikipedia.org/wiki/S-type_asteroid</a> )

### ***WaterType <<enum>>***

An enumeration representing the types of Water that are known. Each enumeration will correspond to it’s own row in a distinct database table.

#### ***Methods***

Method Name	Signature	Description
getWaterType	() : String	Returns the String name of the enumeration property

#### ***Properties***

Property Name	Type	Description
ICE	String	Used by WaterDeposit; waterType is “Ice”
LIQUID	String	Used by WaterDeposit; waterType is “Liquid”
VAPOR	String	Used by WaterDeposit; waterType is “Vapor”
PLASMA	String	Used by WaterDeposit; waterType is “Plasma”

### ***AccessibilityType <<enum>>***

An enumeration representing the types of Accessibility for both Minerals and Water that are found on Asteroids (e.g., minerals might be on the surface, under the surface at a particular depth, on top of a mountain, etc.). Each enumeration will correspond to it’s own row in a distinct database table.

#### ***Methods***

Method Name	Signature	Description
getAccessibilityType	() : String	Returns the String name of the enumeration property



### ***Properties***

Property Name	Type	Description
SURFACE	String	Used by WaterDeposit and MineralDeposit; accessibilityType is "Surface-accessible"
UNDER_SURFACE_SHORT_DEPTH	String	Used by WaterDeposit and MineralDeposit; accessibilityType is "Under surface, short-depth"
UNDER_SURFACE_MEDIUM_DEPTH	String	Used by WaterDeposit and MineralDeposit; accessibilityType is "Under surface, medium-depth"
UNDER_SURFACE_DEEP_DEPTH	String	Used by WaterDeposit and MineralDeposit; accessibilityType is "Under surface, deep-depth"
ABOVE_SURFACE_SHORT_HEIGHT	String	Used by WaterDeposit and MineralDeposit; accessibilityType is "Above surface, short-height"
ABOVE_SURFACE_MEDIUM_HEIGHT	String	Used by WaterDeposit and MineralDeposit; accessibilityType is "Above surface, medium-height"
ABOVE_SURFACE_LARGE_HEIGHT	String	Used by WaterDeposit and MineralDeposit; accessibilityType is "Above surface, large-height"
HEAVY_EXTRACTION_REQUIRED	String	Used by WaterDeposit and MineralDeposit; accessibilityType is "Heavy extraction required; may need additional resources to extract"
UNDER_ICE	String	Used by WaterDeposit and MineralDeposit; accessibilityType is "Under ice"

### ***MineralType <<enum>>***

An enumeration representing the types of known Minerals that may be found on Asteroids and may be mined. Each enumeration will correspond to it's own row in a distinct database table. New types may be added as they are discovered.

### ***Methods***

Method Name	Signature	Description
getMineralType	() : String	Returns the String name of the enumeration property

### ***Properties***

Property Name	Type	Description
GOLD	String	Used by MineralDeposit; mineralType is "Gold"
IRON	String	Used by MineralDeposit; mineralType is "Iron"
IRON_NICKEL	String	Used by MineralDeposit; mineralType is "Iron-Nickel Ore"
PLATINUM	String	Used by MineralDeposit; mineralType is "Platinum"

SILVER	String	Used by MineralDeposit; mineralType is “Silver”
TITANIUM	String	Used by MineralDeposit; mineralType is “Titanium”
PALADIUM	String	Used by MineralDeposit; mineralType is “Paladium”
PLUTONIUM	String	Used by MineralDeposit; mineralType is “Plutonium”
URANIUM	String	Used by MineralDeposit; mineralType is “Uranium”

### ***LifeType <<enum>>***

An enumeration representing the types of known organic Life that may be found on Asteroids. Each enumeration will correspond to it’s own row in a distinct database table. New types may be added as they are discovered.

### ***Methods***

Method Name	Signature	Description
getLifeType	() : String	Returns the String name of the enumeration property

### ***Properties***

Property Name	Type	Description
NONE	String	Used by ExplorationStatus; lifeType is “None”
SINGLE_CELL_ORGANISMS	String	Used by ExplorationStatus; lifeType is “Single-Cell Organisms”
MULTI_CELL_ORGANISMS	String	Used by ExplorationStatus; lifeType is “Multi-Cell Organisms”
INTELLIGENT	String	Used by ExplorationStatus; lifeType is “Intelligent Organisms”
FRIENDLY	String	Used by ExplorationStatus; lifeType is “Friendly Organisms”
HOSTILE	String	Used by ExplorationStatus; lifeType is “Hostile Organisms”

## **Package: cscie97.asn5.spacecraft**

### ***ISpacecraftManagementAPI <<interface>>***

This interface defines the primary service contract for any classes that intend to implement these methods and act as the concrete implementation of the SpacrcraftManagementAPI. All methods require authenticated access via the Authorization Service API; methods must pass a String GUID token, which is checked before continuing. Allows authenticated users

access to manage all aspects of Spacecraft, including creation, listing Spacecraft, searching, and updating Spacecraft. This interface and its implementing class, SpacecraftManagementAPI, collectively work as a façade design pattern - only the public methods here are accessible outside the package.

### **Methods**

Method Name	Signature	Description
getSpacecraft	(String : guid) : List<Spacecraft>	Returns a list of all the Spacecraft under management by the API.
getSpacecraftById	(String : guid, String : id) : Spacecraft	Retrieves the corresponding Spacecraft object from the inventory
updateSpacecraftById	(String : guid, String : spacecraftId, Spacecraft : updatedSpacecraft) : void	Finds the Spacecraft in the inventory with the specified ID, and overwrites that object's properties with those of the updatedSpacecraft. Should also persist the changes to the database.
searchSpacecraft	(String : guid, SpacecraftSearch : searchCriteria) : List<Spacecraft>	Given the supplied SpacecraftSearch criteria object, find and return all Spacecraft that match any of the supplied criteria in the search object.
createNewSpacecraft	(String : guid, Spacecraft : spacecraft) : void	Clients may create new Spacecraft objects independently, but this method must be called to add the Asteroid to the inventory (also persists the newly created Spacecraft to the database).
getSpacecraftTypes	(String : guid) : Set<SpacecraftType>	Returns all the enumeration values of SpacecraftType.
getGuidanceStatuses	(String : guid) : Set<GuidanceStatus>	Returns all the enumeration values of CommunicationStatus.
getCommunicationStatuses	(String : guid) : Set<CommunicationStatus>	Returns all the enumeration values of CommunicationStatus.
getSpacecraftStatuses	(String : guid) : Set<SpacecraftStatus>	Returns all the enumeration values of SpacecraftStatus.
addSpacecraftType	(String : guid, SpacecraftType : type) : void	Adds the new SpacecraftType to the database; subsequent application loads will pick up this new available SpacecraftType.
addGuidanceStatus	(String : guid, GuidanceStatus : type) : void	Adds the new GuidanceStatus to the database; subsequent application loads will pick up this new available GuidanceStatus.
addCommunicationStatus	(String : guid, CommunicationStatus : type) : void	Adds the new CommunicationStatus to the database; subsequent application loads will pick up this new available CommunicationStatus.

### ***SpacecraftManagementAPI, implements ISpacecraftManagementAPI***

This concrete implementation of the ISpacecraftManagementAPI follows the Singleton design pattern; to obtain an instance to use, the getInstance() method will return the sole instance of the object.

#### ***Properties***

Property Name	Type	Description
spacecraft	List<Spacecraft>	Contains all the Spacecraft in the inventory. Spacecraft are initially loaded from the database behind-the-scenes when the application loads and the instance is initialized.
instance <<private>>	SpacecraftManagementAPI	Singleton instance of the API; only one exists at a time

#### ***Methods***

Method Name	Signature	Description
getInstance <<static, synchronized>>	() : ISpacecraftManagementAPI	Used to obtain a Singleton instance of the API for client use.

### ***SpacecraftSearch***

Allows users to create a set of search criteria that richly encapsulates all the properties of a Spacecraft. Since Spacecraft have several numeric and date based properties, utilizes Apache Commons *org.apache.commons.lang3.Range<T>* class to create range-based search criteria (for example, this would allow searching for all Spacecraft with more than 10% fuel remaining but less than 25% remaining, etc.). The search criteria properties are explicitly “OR”, to be as inclusive as possible.

#### ***Properties***

Property Name	Type	Description
rawQuery	String	For text-based searches, simply contains the entire text that the user wishes to search for in String fields on the Spacecraft. May optionally be logged (or saved in the database) to capture a history of searches executed.
launchDate	org.apache.commons.lang3.Range<Date>	Find Spacecraft whose launchDate is between the older and newer Date range

		values passed in the Range<Date> (inclusive)
missionID	String	Find Spacecraft where the ID matches this property
spacecraftType	List<SpacecraftType>	Find all Spacecraft that have a type property value that is one of the passed SpacecraftTypes
fuelPercentageRemaining	org.apache.commons.lang3.Range<Decimal>	Find all Spacecraft that have a fuelPercentageRemaining value on their most recent SpacecraftStatus property that is within the supplied percentage range (inclusive)
guidanceStatus	List<GuidanceStatus>	Find all Spacecraft that have a guidanceStatus value on their most recent SpacecraftStatus property that is one of the supplied GuidanceStatuses
communicationStatus	List<CommunicationStatus>	Find all Spacecraft that have a communicationStatus value on their most recent SpacecraftStatus property that is one of the supplied CommunicationStatuses
status	List<SpacecraftStatus>	Find all Spacecraft that have a currentStatus value on their most recent SpacecraftStatus property that is one of the supplied SpacecraftStatuses
location	org.apache.commons.lang3.Range<Decimal>	Find all Spacecraft that have a location property value that is in between the range supplied (inclusive)
destinationID	String	Find all Spacecraft that have a matching destinationID property; this is a good way to find all the Spacecraft that are heading towards the same Asteroid
spacecraftStatusDate	org.apache.commons.lang3.Range<Date>	Find all Spacecraft that have a statusDate value on any of their SpacecraftStatuses that is between the Date range values (inclusive)

## Spacecraft

Spacecraft model the actual robotic spacecraft that are sent out to explore and mine asteroids. Each Spacecraft contains a history of SpacecraftStatuses that are added autonomously by the spacecraft when out on missions. The real, physical spacecraft that are actually mining asteroids may periodically send a new DiscoveryMessage to the Mission Management System, and at that time the Mission Management System will update the Asteroid where the discovery was made.

### Properties

Property Name	Type	Description
Id	String	Unique identifier for the Spacecraft, and also the primary key in the database.
launchDate	Date	Date that the Spacecraft was launched on its current mission
missionID	String	The unique identifier of the Mission the Spacecraft is currently embarking on
type	SpacecraftType	The type of the Spacecraft
location	Decimal	The current location value of the Spacecraft in AUs
destinationID	String	The unique identifier of the Asteroid where the Spacecraft is headed
statusHistory	List<SpacecraftStatus>	A history of all status updates made to the Spacecraft, including fuel percentage remaining, guidance link status, communications link status, and overall Spacecraft status.
messagesSent	List<DiscoveryMessage>	A list of all DiscoveryMessages that the Spacecraft has made. When the automated robot Spacecraft makes a new discovery of water, minerals, or life, it automatically creates a new DiscoveryMessage and adds it to this list, as well as broadcasting it to the Mission Management System.

### Methods

Method Name	Signature	Description
getCurrentSpacecraftStatus	() : SpacecraftStatus	Returns the most recent SpacecraftStatus based on the date on which it was created.
getSpacecraftType	() : String	Returns the string representation of the SpacecraftType enumeration value.
addNewSpacecraftStatus	(SpacecraftStatus : newStatus) : void	Adds a new SpacecraftStatus to the list. When added the new status should also be persisted to the database.

deleteSpacecraftStatus	(SpacecraftStatus : status) : void	A mechanism to delete an erroneous SpacecraftStatus that may have been added to the Spacecraft. Searches for a matching SpacecraftStatus and deletes it from the Spacecraft if found; also deletes that status record from the database.
sendMessageToMissionControl	(DiscoveryMessage : message) : void	Sends a DiscoveryMessage to the Mission Management System, which is responsible for updating Mission status and Asteroid status.
persist	() : Boolean	Save the Spacecraft and all associated objects (SpacecraftStatus items in statusHistory, and DiscoveryMessage items in messagesSent) to the database; returns true if successful, false otherwise

### ***SpacecraftStatus***

Contains a time-based record of the status of various properties of the Spacecraft at a given point in time. With these objects it is simple to construct a timeline of the overall historical (and current) status of the Spacecraft, which may be useful in investigating root cause analysis when things go wrong. Each record is saved to the database in a table different from the one where Spacecraft records are saved, and has a many-to-one relationship with Spacecraft (one Spacecraft, many SpacecraftStatus records). Any time the parent Spacecraft is saved to the database, the SpacecraftStatus objects are also updated.

### ***Properties***

Property Name	Type	Description
spacecraftID	String	The unique identifier of the associated Spacecraft this status record is for
created	Date	The date and time when this record was first created
fuelPercentageRemaining	Decimal	The amount of fuel remaining in the Spacecraft (expressed as a percentage) at the time the record was created
guidanceStatus	GuidanceStatus	The status of the guidance system of the Spacecraft
communicationLinkStatus	CommunicationStatus	The status of the Spacecraft's communication link system with Mission Control
currentStatus	SpacecraftStatus	A status indicator of the overall health of the Spacecraft and all systems onboard

### ***DiscoveryMessage***

Represents a new discovery of either water, organic life, or minerals on the Asteroid currently being explored by the Spacecraft. This class is virtually identical to the

ExplorationStatus object that is in the Asteroid package; the reason for this duplication is to attempt to isolate the various components as much as possible; the Robotic Spacecraft Management System component should not have a dependency on the Asteroid Inventory System. As the Spacecraft objects explore the Asteroids they are investigating, they will construct these DiscoveryMessages, save the message to the database, and then send the message to the Mission Management System for further processing. DiscoveryMessage objects are stored in their own database table and each one may correspond to a single Spacecraft (Spacecraft can have many associated DiscoveryMessages). Since Spacecraft may have many of these DiscoveryMessages, they may develop a rich history of when things were discovered on the Asteroid, in what quantities, etc.

### ***Properties***

Property Name	Type	Description
id	Integer	A unique identifier for this DiscoveryMessage
spacecraftID	String	The unique identifier of the Spacecraft making the discovery
discoveryDate	Date	When the discovery is made
mineralDeposits	Set<MineralDeposit>	The collection of all Minerals that were found in this discovery
lifeFound	Set<LifeType>	A collection of all the LifeTypes that were found
waterDeposits	Set<WaterDeposit>	A collection of all the WaterDeposits that were found

### ***Methods***

Method Name	Signature	Description
waterFound	() : Boolean	Convenience method to determine if there was any water found in this discovery
lifeFound	() : Boolean	Convenience method to determine if there was any life found in this discovery
mineralsFound	() : Boolean	Convenience method to determine if there was any minerals found in this discovery

### ***SpacecraftType <<enum>>***

An enumeration representing the types of Spacecraft currently in use. Each enumeration will correspond to it's own row in a distinct database table.

### ***Methods***

Method Name	Signature	Description
getSpacecraftType	() : String	Returns the String name of the enumeration property



### ***Properties***

Property Name	Type	Description
EXPLORER	String	Used by Spacecraft; spacecraftType is “Explorer”
MINER	String	Used by Spacecraft; spacecraftType is “Miner”
SUPPORT_CRAFT	String	Used by Spacecraft; spacecraftType is “Support Craft”
REPAIR_CRAFT	String	Used by Spacecraft; spacecraftType is “Repair Craft”
REFUELER	String	Used by Spacecraft; spacecraftType is “Refueler”
HEAVY_EXCAVATOR	String	Used by Spacecraft; spacecraftType is “Heavy Excavator”
TOW_CRAFT	String	Used by Spacecraft; spacecraftType is “Tow Craft”

### ***GuidanceStatus <<enum>>***

An enumeration representing the status values for the Spacecraft guidance system. Each enumeration will correspond to it’s own row in a distinct database table.

### ***Methods***

Method Name	Signature	Description
getGuidanceStatus	() : String	Returns the String name of the enumeration property

### ***Properties***

Property Name	Type	Description
ALL_OK	String	Used by SpacecraftStatus; guidanceStatus is “All-OK”
SATELLITE_UPLINK_DOWN	String	Used by SpacecraftStatus; guidanceStatus is “Satellite Uplink Down”
SYSTEM_FAILURE	String	Used by SpacecraftStatus; guidanceStatus is “Guidance System Failure”
POWER_FAILURE	String	Used by SpacecraftStatus; guidanceStatus is “Guidance System Power Failure”
FUSE_BLOWN	String	Used by SpacecraftStatus; guidanceStatus is “Guidance System Fuse Blown”
SOLAR_PANEL_FAILURE	String	Used by SpacecraftStatus; guidanceStatus is “Guidance Sytstem Solar Panel Failure”

### ***CommunicationStatus <<enum>>***

An enumeration representing the status values for the Spacecraft's communications system. Each enumeration will correspond to it's own row in a distinct database table.

### **Methods**

Method Name	Signature	Description
getCommunicationStatus	() : String	Returns the String name of the enumeration property

### **Properties**

Property Name	Type	Description
ALL_OK	String	Used by SpacecraftStatus; communicationStatus is "All-OK"
COMM_TRANSMISSION_FAILURE	String	Used by SpacecraftStatus; communicationStatus is "Communication System Transmission Failure"
SIGNAL_INTERFERENCE	String	Used by SpacecraftStatus; communicationStatus is "Communication System Signal Interference"
ANTENNA_DOWN	String	Used by SpacecraftStatus; communicationStatus is "Communication System Antenna Down"
SIGNAL_RELAY_MODE	String	Used by SpacecraftStatus; communicationStatus is "Communication System Signal Relay Mode" (sends signals to other nearby Spacecraft and asks them to send message on Spacecraft's behalf to Mission Control)

### **SpacecraftStatusType <<enum>>**

An enumeration representing the overall mission status values for the Spacecraft. Each enumeration will correspond to it's own row in a distinct database table.

### **Methods**

Method Name	Signature	Description
getSpacecraftStatusType	() : String	Returns the String name of the enumeration property

### **Properties**

Property Name	Type	Description
AWAITING_LAUNCH	String	Used by SpacecraftStatus; spacecraftStatusType is "Waiting For Launch"
EN_ROUTE_TO_ASTEROID	String	Used by SpacecraftStatus; spacecraftStatusType is "En-Route To Asteroid"
LOST	String	Used by SpacecraftStatus; spacecraftStatusType is "Lost"
CRASHED	String	Used by SpacecraftStatus; spacecraftStatusType is "Crashed"

LANDED	String	Used by SpacecraftStatus; spacecraftStatusType is “Landed”
EXPLORING	String	Used by SpacecraftStatus; spacecraftStatusType is “Exploring”
MINING	String	Used by SpacecraftStatus; spacecraftStatusType is “Mining”
REFUELING	String	Used by SpacecraftStatus; spacecraftStatusType is “REFUELING”
MALFUNCTION	String	Used by SpacecraftStatus; spacecraftStatusType is “Malfunction”
HOMEWARD_BOUND	String	Used by SpacecraftStatus; spacecraftStatusType is “Homeward Bound”

The cscie97.asn5.spacecraft package also has exact copies of several enumeration types and classes from the cscie97.asn5.asteroid package, including:

- MineralDeposit <<class>>
- WaterDeposit <<class>>
- MineralType <<enum>>
- AccessibilityType <<enum>>
- WaterType <<enum>>
- LifeType <<enum>>

For sake of brevity, the definitions of these types is not replicated here (they are exact copies of those already defined in the cscie97.asn5.asteriod package).

## Package: cscie97.asn5.mission

### *IEarthBasedCommunicationLink <<interface>>*

This interface defines the primary service contract for any classes that intend to implement these methods and act as the concrete implementation of the EarthBasedCommunicationLink. This class handles processing incoming DiscoveryMessages from Spacecraft, and also maintains a Communication Link Status property to indicate if it is functional and still able to receive messages from Spacecraft.

### **Methods**

Method Name	Signature	Description
receiveIncomingMessage	(cscie97.asn5.spacecraft.DiscoveryMessage : message) : void	Provides a mechanism for Spacecraft to send messages when they discover new resources on an Asteroid.
communicationLinkActive	() : Boolean	Checks whether or not the current communication link

		is active and able to receive messages from Spacecraft
--	--	--

### ***EarthBasedCommunicationLink, implements IEarthBasedCommunicationLink***

This concrete implementation of the IEarthBasedCommunicationLink, follows the Singleton design pattern; to obtain an instance to use, the getInstance() method will return the sole instance of the object. All messages that are received should be stored in a persistent database table as they arrive.

#### ***Properties***

Property Name	Type	Description
messagesReceived	List<cscie97.asn5.spacecraft.DiscoveryMessage>	A list of all the DiscoveryMessages that have been received from Spacecraft in the field.
commLinkActive	Boolean	Denotes if the communication link is “active” and able to receive incoming messages from Spacecraft
Instance <<private>>	EarthBasedCommunicationLink	A private Singleton instance of EarthBasedCommunicationLink, used by all clients

#### ***Methods***

Method Name	Signature	Description
getInstance <<static, synchronized>>	() : ISpacecraftManagementAPI	Used to obtain a Singleton instance of the API for client use.

### ***InventoryManager <<interface>>***

This interface defines the primary service contract for any classes that intend to implement these methods and act as the concrete implementation of the InventoryManager. This class provides getter methods for obtaining several key pieces of information about the overall program, including overall program budget, numbers of Spacecraft “in the field” and number docked on Earth, and the amount of fuel available for all Spacecraft to use.

#### ***Methods***

Method Name	Signature	Description
getTotalOperatingBudget	() : Decimal	Returns the overall budget available to the entire Asteroid Exploration Program, minus the cost of currently running Missions; can be used to prohibit

		a new Mission from beginning if the Mission cost is more than the available budget.
getNumberOfSpacecraftInUse	() : Integer	Returns the number of Spacecraft “in the field”
getNumberOfSpacecraftNotInUse	() : Integer	Returns the number of Spacecraft that are currently docked and thus available for use in new Missions
getAmountFuelAvailable	() : decimal	Returns the amount of fuel currently available to use for any new Missions
getFuelForMission	(Decimal : amountToUse) : void	Deducts the amount of fuel requested for a Mission from the overall total amount of fuel available. Should throw a custom exception if the amount requested exceeds the amount available.
requisitionSpacecraftForMission	(List<String> : spacecraftIdsTouse) : void	Requisitions the spacecraft specified by their IDs for a new Mission. Updates the status of those spacecraft and prepares them for mission launch; updates the number of spacecraft in-use/not-in-use.
returnSpacecraft	(String : spacecraftId) : void	Returns a spacecraft back into the stable and makes it available for future missions; updates the number of spacecraft in-use/not-in-use.

### ***InventoryManager, implements IInventoryManager***

This concrete implementation of the IInventoryManager follows the Singleton design pattern; to obtain an instance to use, the getInstance() method will return the sole instance of the object. All basic properties should be stored in a persistent database table as they are created and updated.

### ***Properties***

Property Name	Type	Description
totalAvailableOperatingBudget	Decimal	The overall remaining budget figure that is to fund all future missions.
numberSpacecraftInUse	Integer	A count of the number of Spacecraft currently “in the field”
numberSpacecraftNotInUse	Integer	A count of the number of Spacecraft currently docked on Earth
amountFuelAvailable	Decimal	The amount of Spacecraft fuel ready for use on Earth, expressed in tons
instance <<private>>	IInventoryManager	A private Singleton instance of IInventoryManager, used by all clients

### ***Methods***

Method Name	Signature	Description
getInstance <<static, synchronized>>	() : InventoryManager	Used to obtain a Singleton instance of the API for client use.

### ***IMissionManagementAPI <<interface>>***

This interface defines the primary service contract for any classes that intend to implement these methods and act as the concrete implementation of the MissionManagementAPI. The implementing class acts as the primary broker of communication from Spacecraft, as well as bears the responsibility for updating Asteroids and their ExplorationStatuses as new discoveries are made. It also administers Missions, creates them, updates them, and interacts with the InventoryManager for overall mission control. All methods (except incoming DiscoveryMessages from Spacecraft) requires authentication, and so must pass a GUID string that the AuthenticationService will validate.

### ***Methods***

Method Name	Signature	Description
getMissions	(String : guid) : List<Mission>	Returns all the Missions in the system.
getMissionById	(String : guid, String : id) : Mission	Returns the specific Mission that matches the ID passed.
updateMissionById	(String : guid, String : missionId, Mission : updatedMission) : void	Updates the Mission whose ID matches the missionID passed, and updates all it's values with those from the updatedMission. Writes the updated Mission to the database.
searchMissions	(String : guid, MissionSearch : searchCriteria) : List<Mission>	Searches the inventory of Missions for those that match any of the supplied criteria in the searchCriteria, and returns them as a list.
createNewMission	(String : guid, Mission : mission) : void	Creates a new Mission, and saves it to the database. If a Mission may not be created because it violates the budgetary requirements, not enough Spacecraft are available, etc., then an Exception is thrown.
getMissionStatuses	(String : guid) : Set<MissionStatus>	Returns the list of all MissionStatus enumeration types.

addMissionStatus	(String : guid, MissionStatus : type) : void	Adds a new enumeration type to the MissionStatus, and saves it to the database so that an application restart will make it available.
beginMission	(String : guid, Mission : mission) : void	Begin a Mission and update the statuses of Spacecraft.
updateMissionResources	(cscie97.asn5.spacecraft.DiscoveryMessage : message) : void	Update the Mission resources based on the DiscoveryMessage received from a Spacecraft

### ***MissionManagementAPI, implements IMissionManagementAPI***

This concrete implementation of the IMissionManagementAPI follows the Singleton design pattern; to obtain an instance to use, the getInstance() method will return the sole instance of the object. This is the central public administration interface for Missions.

#### ***Properties***

Property Name	Type	Description
missions	List<Mission>	The inventory of all Missions, past and present.
inventoryManager	IIventoryManager	An instance of the InventoryManager, to update resources for all Missions as appropriate.
instance <<private>>	IMissionManagementAPI	A private Singleton instance of IIventoryManager, used by all clients

#### ***Methods***

Method Name	Signature	Description
getInstance <<static, synchronized>>	() : IMissionManagementAPI	Used to obtain a Singleton instance of the API for client use.

### ***MissionSearch***

Allows users to create a set of search criteria that richly encapsulates all the properties of a Mission and use that for searching. Since Missions have several numeric and date based properties, utilizes Apache Commons *org.apache.commons.lang3.Range<T>* class to create range-based search criteria (for example, this would allow searching for all Missions whose Spacecraft have more than 10% fuel remaining but less than 25% remaining, etc.). The search criteria properties are explicitly “OR”, to be as inclusive as possible.

#### ***Properties***

Property Name	Type	Description
rawQuery	String	For text-based searches, simply contains the entire text that the user wishes to search for in String fields on the Mission. May optionally be logged (or saved in the database) to capture a history of searches executed.
launchDate	org.apache.commons.lang3.Range<Date>	Find Missions whose launchDate is between the older and newer Date range values passed in the Range<Date> (inclusive)
missionID	String	Find Missions where the ID matches this property
spacecraftType	List<SpacecraftType>	Find all Misisions with Spacecraft that has a type property value that is one of the passed SpacecraftTypes
fuelPercentageRemaining	org.apache.commons.lang3.Range<Decimal>	Find all Missions with Spacecraft that have a fuelPercentageRemaining value on their most recent SpacecraftStatus property that is within the supplied percentage range (inclusive)
guidanceStatus	List<GuidanceStatus>	Find all Missions with Spacecraft that have a guidanceStatus value on their most recent SpacecraftStatus property that is one of the supplied GuidanceStatuses
communicationStatus	List<CommunicationStatus>	Find all Missions with Spacecraft that have a communicationStatus value on their most recent SpacecraftStatus property that is one of the supplied CommunicationStatuses
status	List<SpacecraftStatus>	Find all Missions with Spacecraft that have a currentStatus value on their most recent SpacecraftStatus property that is one of the supplied SpacecraftStatuses
location	org.apache.commons.lang3.Range<Decimal>	Find all Missions with Spacecraft that have a location property value that is



		in between the range supplied (inclusive)
destinationID	String	Find all Missions with Spacecraft that have a matching destinationID property; this is a good way to find all the Missions that are heading towards the same Asteroid

## ***Mission***

Missions model the exploratory efforts and Spacecraft that are sent out to Asteroids for exploration and mining of resources. Missions may be comprised of multiple Spacecraft, or just a single Spacecraft, depending on the size of the Asteroid destination. Missions should be stored in a database table and use the unique ID property as the database key; calling the “persist()” method should write the current objects to the database.

## ***Properties***

Property Name	Type	Description
Id	String	Unique identifier for the Mission, and also the primary key in the database.
name	String	The name of the Mission.
purpose	String	Explanation of the purpose for this Mission.
spacecraftIDs	List<String>	The list of Spacecraft IDs that are going on this Mission.
launchDate	Date	Date that the Mission was officially launched
estimatedTimeArrival	Date	Estimated time that the Spacecraft will reach the destination Asteroid
destinationAsteroidID	String	The ID of the Asteroid the Mission is heading to
status	MissionStatus	The current overall status of the Mission
missionOperatingBudget	Decimal	The budget for the entire Mission
notes	List<MissionNotes>	Notes that have been entered for the Mission by mission control administrators.

## ***Methods***

Method Name	Signature	Description
addSpacecraftToMission	(String : id) : void	Adds the Spacecraft with matching ID to the list of Spacecraft that are going on this Mission.
removeSpacecraftFromMission	(String : id) :	Removes the Spacecraft with matching ID from the list of

	void	Spacecraft participating in the Mission.
persist	() : Boolean	Save the Mission and all associated objects (MissionNotes items in notes) to the database; returns true if successful, false otherwise

### ***MissionNotes***

Contains a time-based record of notes that mission control administrators have made on the associated Mission. With these objects it is simple to construct a timeline of the overall historical notes that have been added to the Mission, which may be useful in post-mortem analysis of Mission success or failure. Each record is saved to the database in a table different from the one where Mission records are saved, and has a many-to-one relationship with Mission (one Mission, many MissionNotes records). Any time the parent Mission is saved to the database, the MissionNotes objects are also updated.

### ***Properties***

Property Name	Type	Description
dateAdded	Date	When the notes were created
notes	String	The content of the mission notes

### ***MissionStatus <<enum>>***

An enumeration representing the various MissionStatuses that a given Mission may fall under. Each enumeration will correspond to it's own row in a distinct database table.

### ***Methods***

Method Name	Signature	Description
getMissionStatus	() : String	Returns the String name of the enumeration property

### ***Properties***

Property Name	Type	Description
IN_PROGRESS	String	Used by Mission; missionStatus is "In-Progress"
ABORTED	String	Used by Mission; missionStatus is "Aborted"
COMPLETED	String	Used by Mission; missionStatus is "Completed"
PLANNED	String	Used by Mission; missionStatus is "Planned"

**Package: cscie97.asn5.authentication**

The Authentication sub-system should follow the exact same design and implementation as the IAuthenticationServiceAPI used in Assignment 4. To see the design for the Authentication System, refer to the design submission for Assignment 4.

## Sequence Diagrams

// TODO: make the following sequence diagrams:

- Message flow for receiving status message from a spacecraft where spacecraft has discovered water on target asteroid
- Message flow for receiving a status message where the spacecraft has completed its mission
- Message flow for provisioning a new mission

## Command and Control User Interface Wireframes

In order to support mission control administrators, a web interface for the Asteroid Exploration System should be built that includes areas for allowing administrators to do the following:

- Log in and Log Out of the system
- Define new Missions
- Monitoring and updating Mission status
- Monitoring and updating Spacecraft status
- Monitoring and updating Asteroid status
- Monitoring Mission Control resources (fuel, number of available spacecraft, operating budget)
- Monitoring status of the ground based communication links
- Monitoring incoming DiscoveryMessages from the Spacecraft

The following series of wireframe images illustrate how the system should be visually partitioned and explains through side notes how the user interactions should work.

Asteroid Exploration System : Login

http://nasa.gov/cscie97/asteroidexploration/login

# Asteroid Exploration System

## Welcome

[Home](#) > ...

Please enter your credentials to use the system. If you've forgotten your username or password, contact [Central Control](#) to re-establish your system identity and credentials.

Username  Password

[Home](#) | [Missions](#) | [Spacecraft](#) | [Asteroids](#) | [Resources](#) | [Communications](#)

users enter their credentials here to authenticate and login


Asteroid Exploration System : Missions

http://nasa.gov/cscie97/asteroidexploration/missions

# Asteroid Exploration System

## Mission Central

[Home](#) > Missions Welcome, [dkilleffer](#)





 Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus libero sapien, facilisis adipiscing odio a, vestibulum ultrices lectus. Nulla eget enim a eros posuere mattis at et nibh. In hac habitasse platea dictumst. Phasellus et leo at enim tincidunt egestas a at tortor. Praesent facilisis nisi non malesuada tempor. Integer libero

primary page where mission based tasks are initiated


[Define New Mission](#)  
[View Current Missions](#)  
[View Past Missions](#)  
[Monitor & Update Mission Status](#)

[Home](#) | [Missions](#) | [Spacecraft](#) | [Asteroids](#) | [Resources](#) | [Communications](#)

Asteroid Exploration System : Missions : Define New Mission



http://nasa.gov/cscie97/asteroidexploration/missions/define




# Asteroid Exploration System

## Mission Central

[Home](#) > [Missions](#) > Define New Mission

Welcome, [dkilleffer](#).



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus libero sapien, facilisis adipiscing odio a, vestibulum ultrices lectus. Nulla eget enim a eros posuere mattis at et nibh. In hac habitasse platea dictumst. Phasellus et leo at enim tincidunt egestas a at tortor. Praesent facilisis nisi non malesuada tempor. Integer libero

Users can create new missions here and define the major aspects of the mission

Mission ID

Mission Name

Purpose

Estimated Budget

Intended Launch Date

Estimated Arrival Date

Destination Asteroid

/ /

/ /

[select asteroid destination](#)

1 Ceres

4 Vesta

433 Eros

etc.

Select Available Spacecraft To Support Mission

☐ Voyager (type: miner, cargo capacity: 2 tons)

☒ Radiance (type: miner, cargo capacity: 1.5 tons)

☒ Vigilance (type: refueler, fuel capacity: 6000 liters)

☒ Hyperion (type: heavy excavator)

☒ Xeroen (type: materials analysis)

☒ Meroleseum (type: support & repair, towing capacity: 1 ton)

☐ Hercules (type: miner, capacity: 6 tons) NOTE: undergoing maintenance





☐ Zaren (will be used by different mission during selected launch dates)

Clear Form


Submit New Mission Request

[Home](#) | [Missions](#) | [Spacecraft](#) | [Asteroids](#) | [Resources](#) | [Communications](#)

Asteroid Exploration System : Missions : Update Mission Status



http://nasa.gov/cscie97/asteroidexploration/missions/update




# Asteroid Exploration System

## Mission Central





[Home](#) > [Missions](#) > Update Mission Status

Welcome, [dkilleffer](#)



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus libero sapien, facilisis adipiscing odio a, vestibulum ultrices lectus. Nulla eget enim a eros posuere mattis at et nibh. In hac habitasse platea dictumst. Phasellus et leo at enim tincidunt egestas a at tortor. Praesent facilisis nisi non malesuada tempor. Integer libero

### Status Flags



Mission ID

explore-ceres-1-water

Mission Name

Explore Ceres 1 Asteroid for Water Resources

Purpose


The primary purpose of this mission is to discover whether or not the Ceres-1 asteroid contains any water resources in any form.

Actual Budget

\$975,000,000


Actual Launch Date

2018/06/03



Actual Arrival Date

2018/12/23



Destination Asteroid

1Ceres

Deployed Spacecraft

☒ Radiance (type: miner, cargo capacity: 15 tons)

☒ Vigilance (type: refueler, fuel capacity: 6000 liters)

☒ Hyperion (type: heavy excavator)

☒ Xeroen (type: materials analysis)

☒ Meroleseum (type: support & repair, towing capacity: 1 ton)

Update Status

select new mission status

Waiting For Launch

In Progress

Complete

Aborted

Cancelled

admins may change the mission status here

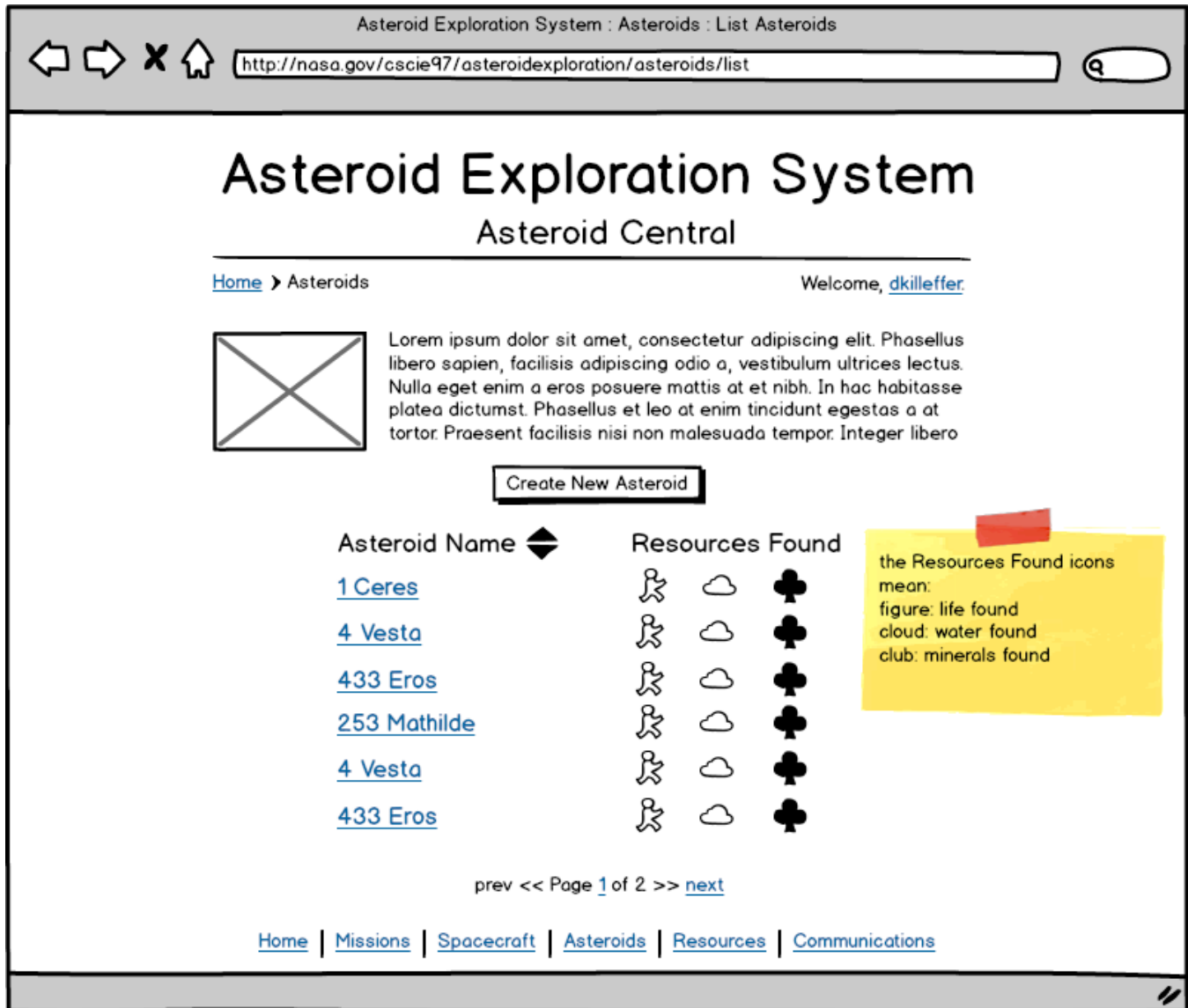
these fields are not editable now

Clear Form





Submit Mission Update

[Home](#) | [Missions](#) | [Spacecraft](#) | [Asteroids](#) | [Resources](#) | [Communications](#)

46



Asteroid Exploration System : Spacecraft : List Spacecraft




http://nasa.gov/cscie97/asteroidexploration/spacecraft/list

Q

# Asteroid Exploration System


## Spacecraft Central

[Home](#) > [Spacecraft](#)Welcome, [dkilleffer](#).

































Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus libero sapien, facilisis adipiscing odio a, vestibulum ultrices lectus. Nulla eget enim a eros posuere mattis at et nibh. In hac habitasse platea dictumst. Phasellus et leo at enim tincidunt egestas a at tortor. Praesent facilisis nisi non malesuada tempor. Integer libero

Create New Spacecraft

Spacecraft Name 

[Voyager \(type: miner, cargo capacity: 2 tons\)](#)  
[Radiance \(type: miner, cargo capacity: 1.5 tons\)](#)  
[Vigilance \(type: refueler, fuel capacity: 6000...\)](#)  
[Hyperion \(type: heavy excavator\)](#)  
[Xeroen \(type: materials analysis\)](#)  
[Meroleseum \(type: support & repair, towing...\)](#)

Indicators

prev << Page 1 of 2 >> [next](#)

[Home](#) | [Missions](#) | [Spacecraft](#) | [Asteroids](#) | [Resources](#) | [Communications](#)

the indicator icons mean:  
battery: indicates approximate power remaining  
vertical bars: displays health of radio communication strength  
exclamation mark: warning, status problem with spacecraft  
lightning bolt: indicates a problem is present with radio transmission/reception  
globe: indicates spacecraft is returning to earth



← → × 🏠

Asteroid Exploration System : Spacecraft : Update Spacecraft Status

http://nasa.gov/cscie97/asteroidexploration/spacecraft/update

🔍

# Asteroid Exploration System

## Spacecraft Central

[Home](#) > [Spacecraft](#) > Update Spacecraft Status

Welcome, [dkilleffer](#)

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus libero sapien, facilisis adipiscing odio a, vestibulum ultrices lectus. Nulla eget enim a eros posuere mattis at et nibh. In hac habitasse platea dictumst. Phasellus et leo at enim tincidunt egestas a at tortor. Praesent facilisis nisi non malesuada tempor. Integer libero

### Indicator Flags

admins may view all current spacecraft detailed info here, as well as browse past status updates

Spacecraft ID

radiance-111

Spacecraft Name

Radiance X-111 Prototype

Type

Miner

Type Details

cargo capacity: 1.5 tons

Current Mission

[Explore Ceres 1 Asteroid for Water Resources](#)

Destination Asteroid

[Ceres 1](#)

Actual Launch Date

2018/06/03

these fields are not editable now

### Past Status Updates Log

Log Created	Fuel % Remain	Guidance Status	Comm. Status	Current Status
2125-11-02	78%	✓	✓	✓
2125-11-03	77%	✓	✓	✓
2125-11-04	76%	✓	✓	✓
2125-11-05	75%	✓	✓	✓

### Append Status Log Entry

Log Created	Fuel % Remain	Guidance Status	Comm. Status	Current Status
<div>//</div>	<div>%</div>	<div>enter status</div>	<div>enter status</div>	<div>enter status</div>

Clear Form

Submit Status Update

admins may append to the spacecraft status log here

[Home](#) | [Missions](#) | [Spacecraft](#) | [Asteroids](#) | [Resources](#) | [Communications](#)

49

Asteroid Exploration System : Spacecraft : Create New Spacecraft

http://nasa.gov/cscie97/asteroidexploration/spacecraft/create


Q

# Asteroid Exploration System

## Spacecraft Central

[Home](#) > [Spacecraft](#) > Create New Spacecraft

Welcome, [dkilleffer](#).



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus libero sapien, facilisis adipiscing odio a, vestibulum ultrices lectus. Nulla eget enim a eros posuere mattis at et nibh. In hac habitasse platea dictumst. Phasellus et leo at enim tincidunt egestas a at tortor. Praesent facilisis nisi non malesuada tempor. Integer libero

Spacecraft ID

Spacecraft Name

Type

select type

Miner

Refueler

Materials Analysis

Heavy Excavator

Support & Repair

Surface Mapper

Type Details

type-specific-details-loaded-here

Destination Asteroid

select asteroid

Ceres 1

4 Vesta

433 Eros

...

Expected Launch Date

/ /

Add Status Log Entry

Log Created

/ /

Fuel % Remain

%

Guidance Status

enter status

Comm. Status

enter status

Current Status

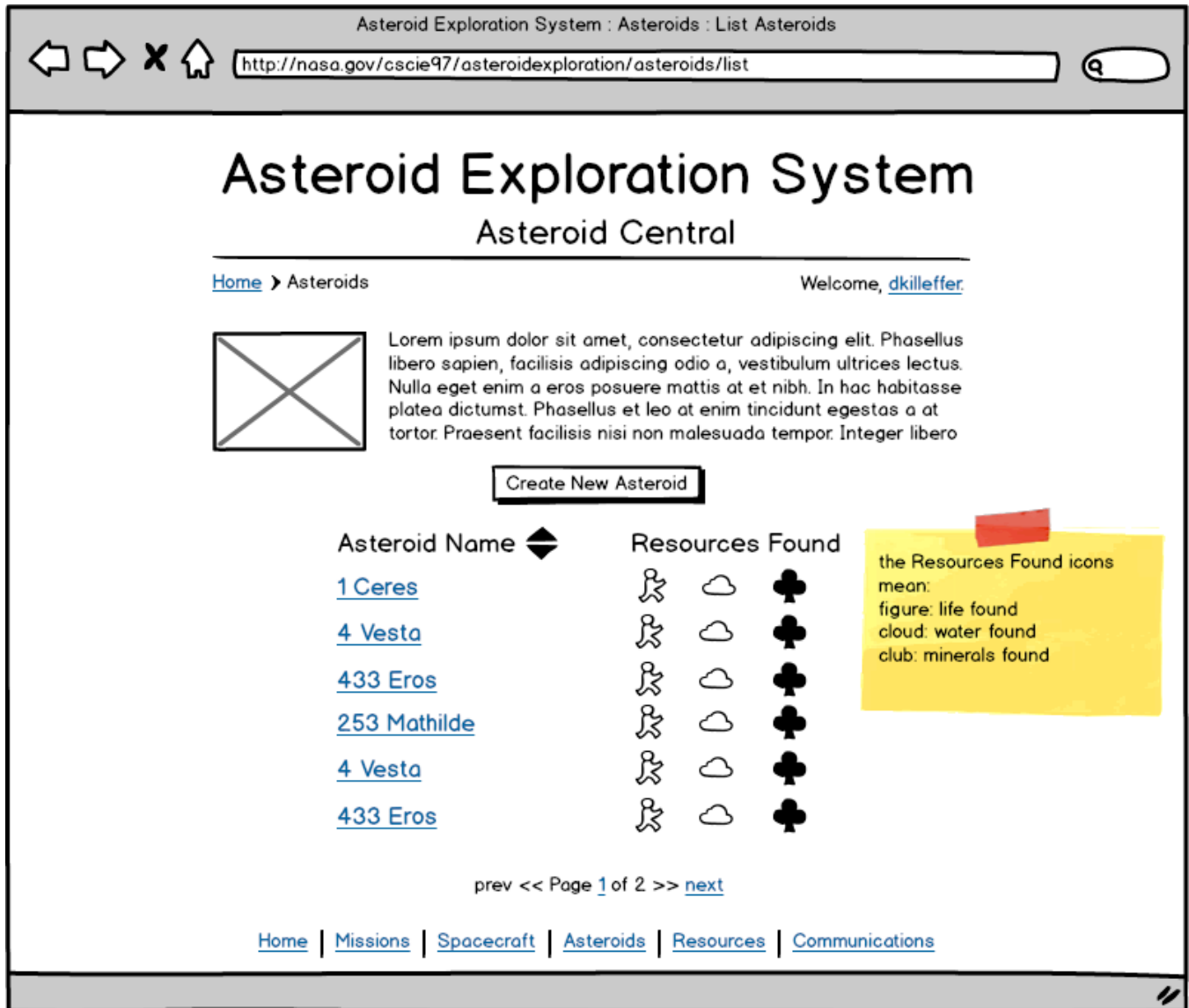
enter status

Clear Form

Create Spacecraft

[Home](#) | [Missions](#) | [Spacecraft](#) | [Asteroids](#) | [Resources](#) | [Communications](#)

admins may  
append to the  
spacecraft  
status log here



←

→

✕

🏠

Asteroid Exploration System : Asteroids : Update Asteroid Info

http://nasa.gov/cscie97/asteroidexploration/asteroid/update


🔍

# Asteroid Exploration System

## Asteroid Central

[Home](#) > [Asteroids](#) > Update Asteroid Info

Welcome, [dkilleffer](#).



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus libero sapien, facilisis adipiscing odio a, vestibulum ultrices lectus. Nulla eget enim a eros posuere mattis at et nibh. In hac habitasse platea dictumst. Phasellus et leo at enim tincidunt egestas a at tortor Praesent facilisis nisi non malesuada tempor. Integer libero

admins may view all current asteroid detailed info here, as well as browse past notes updates and exploration discoveries (minerals, life,

Asteroid ID

253-mathilde

Asteroid Name

253 Mathilde

Type

C-type

Primary Notes

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

Width

25.5

Height

25.5

Length

25.5

Surface Gravity

0.00345

Mass

98424.5

Aphelion

5834

Perihelion

12324

Historical Notes

Note Date	Note Author	Notes
2125-11-02	dkilleffer	Lorem ipsum dolor sit amet, consectetur adipiscing elit.
2125-11-03	dkilleffer	Lorem ipsum dolor sit amet, consectetur adipiscing elit.
2125-11-04	dkilleffer	Lorem ipsum dolor sit amet, consectetur adipiscing elit.
2125-11-05	dkilleffer	Lorem ipsum dolor sit amet, consectetur adipiscing elit.

Past Exploration Status Updates

Date	Minerals	Life	Water
2125-11-02	Zinc, 10 tons		
2125-11-03		bacterial, in ice	5 trillion gallons, ice
2125-11-04	Tungsten, 0.25 tons		
2125-11-05	Copper, 9K tons		

Append Note

New Discovery Details

Mineral Type

enter mineral type

Copper

Zinc

Tungsten

Iron

Mineral Accessibility

enter accessibility

Buried > 3 miles deep

Surface Level

> 1 mile above ground

Encased in Ice

Water

water type

Liquid

Ice

Vapor

Plasma

Life

enter life

Microbial

Bacterial

Humanoïd

Hostile

Friendly

Advanced

Clear Form

Submit Asteroid Update

these fields are not editable now

when appending a new note, the system automatically detects the current user and time and uses those values

[Home](#) | [Missions](#) | [Spacecraft](#) | [Asteroids](#) | [Resources](#) | [Communications](#)

←

→

✕

🏠

Asteroid Exploration System : Asteroids : Search For Asteroids

http://nasa.gov/cscie97/asteroidexploration/asteroid/search


Q

# Asteroid Exploration System

## Asteroid Central

[Home](#) > [Asteroids](#) > Search For Asteroids

Welcome, [dkilleffer](#).



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus libero sapien, facilisis adipiscing odio a, vestibulum ultrices lectus. Nulla eget enim a eros posuere mattis at et nibh. In hac habitasse platea dictumst. Phasellus et leo at enim tincidunt egestas a at tortor. Praesent facilisis nisi non malesuada tempor. Integer libero

admins may search for all asteroids that match any of the entered criteria here; similar searches are available for both Missions and Spacecraft (though separate search wireframes for Missions and Spacecraft isn't shown)

Asteroid Search Terms

Asteroid Types

☒ A-Type

☒ L-Type

☒ R-Type

☒ C-Type

☒ M-Type

☒ S-Type

☒ K-Type

☒ Q-Type

☒ InnerbeltComet

Min. Width

0

Max. Width

qqqqqqq

Min. Length

0

Max. Length

qqqqqqq

Min. Surface Gravity

0.00000

Max. Surface Gravity

qqqqqqq

Min. Mass

0.00000

Max. Mass

qqqqqqq

Min. Aphelion

0.00000

Max. Aphelion

qqqqqqq

Min. Perihelion

0.00000

Max. Perihelion

qqqqqqq

Minerals Found

☒ Gold

☒ Platinum

☒ Paladium

☒ Iron

☒ Silver

☒ Plutonium

☒ Iron-Nickel Ore

☒ Titanium

☒ Uranium

Life Found

☒ None

☒ Intelligent

☒ Single-Cell Organisms

☒ Friendly

☒ Multi-Cell Organisms

☒ Hostile

Water Found

☒ Ice

☒ Liquid

☒ Vapor





☐ Plasma

Clear Form


Search Asteroids

[Home](#) | [Missions](#) | [Spacecraft](#) | [Asteroids](#) | [Resources](#) | [Communications](#)

Asteroid Exploration System : Resources : View Resource Info



http://nasa.gov/cscie97/asteroidexploration/resources/view




# Asteroid Exploration System

## Resources Central

[Home](#) > Resources

Welcome, [dkilleffer](#).



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus libero sapien, facilisis adipiscing odio a, vestibulum ultrices lectus. Nulla eget enim a eros posuere mattis at et nibh. In hac habitasse platea dictumst. Phasellus et leo at enim tincidunt egestas a at tortor. Praesent facilisis nisi non malesuada tempor. Integer libero

admins may view and edit most of the resources (except for number of spacecraft in use) here

Total Available Budget \$

\$289,500,000,000

Number Spacecraft In Use

47

} this field not editable here

Number Spacecraft Available For Missions

13

Fuel Available

79342


tons

Clear Form

Submit Resources Update

[Home](#) | [Missions](#) | [Spacecraft](#) | [Asteroids](#) | [Resources](#) | [Communications](#)

Asteroid Exploration System : Communications : View Communications Status



http://nasa.gov/cscie97/asteroidexploration/communications/status


Q

# Asteroid Exploration System

## Communications Central

[Home](#) > Communications

Welcome, [dkilleffer](#).



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus libero sapien, facilisis adipiscing odio a, vestibulum ultrices lectus. Nulla eget enim a eros posuere mattis at et nibh. In hac habitasse platea dictumst. Phasellus et leo at enim tincidunt egestas a at tortor. Praesent facilisis nisi non malesuada tempor. Integer libero

### Current Earth-Based Communication Link Status

✓  
Good!

### Recent Incoming Messages Received From Spacecraft

Date	Mission	Spacecraft	Minerals	Life	Water
2125-11-02	<a href="#">Explore Ceres 1...</a>	<a href="#">Voyager</a>	Zinc, 10 tons		
2125-11-03	<a href="#">Explore Ceres 1...</a>	<a href="#">Voyager</a>		bacterial, in ice	5 trillion gallons, ice
2125-11-04	<a href="#">Explore Ceres 1...</a>	<a href="#">Voyager</a>	Tungsten, 0.25 tons		
2125-11-05	<a href="#">Explore Ceres 1...</a>	<a href="#">Voyager</a>	Copper, 9K tons		

[Home](#) | [Missions](#) | [Spacecraft](#) | [Asteroids](#) | [Resources](#) | [Communications](#)

## Implementation Details

Several classes in the Asteroid Exploration System are to be implemented as Singletons; examples are the AsteroidInventoryAPI, SpacecraftManagementAPI, and MissionManagementAPI. Additionally, these public APIs are also implemented as instances of their service interface definitions (IAsteroidInventoryAPI, ISpacecraftManagementAPI, and IMissionManagementAPI, respectively). This follows the Façade design pattern, and only exposes the public interface of the services to outside clients. Anywhere in the code that a reference to any of the service implementations is expected, the interface type is returned rather than a concrete implementation.



For data persistence, an Object-Relational Mapping software system should be used to directly map objects (such as Asteroid, Mission, Spacecraft, ExplorationStatus, etc.) to database tables. Depending on the implementing language, if Java is selected as the implementation language and platform, Hibernate is a good choice for an ORM layer, but there are several others to choose from. For the purposes of this sample design, the objects themselves may both be directly stored in the database as well as returned to clients when requesting objects; normally the underlying objects would not be returned to clients directly, but a Data Transfer Object (DTO) would be instead.

All Enumeration types should be stored in their own database tables, and when the application is loaded, they should be read from the database and populate the potential enumeration values. The requirements for this design did not call for an Entity-Relationship diagram of the intended database layout to be included, but in general, enumeration types should be short, small tables with one row per enumeration value. For objects that contain enumeration type values (for example, ExplorationStatus in the cscie97.asn5.asteroid package), the ExplorationStatus objects will require their own database table, but another join table will be necessary to track each enumeration type that the ExplorationStatus is meant to contain multiples of (for example, one record in the table where ExplorationStatuses are saved will correspond to N-records in another table where the ExplorationStatus' MineralTypes are meant to be stored).

## Changes from Original Design & Requirements

The original requirements for the Asteroid Exploration System only called for including one type of search functionality; the ability to search for Asteroids that match a particular text search criteria, and only searching within the Notes saved on each Asteroid. I felt that a much richer search would be expected by users of such a system in practice, and that any property of an Asteroid should be a valid search vector. As such, I designed a much richer search object (AsteroidSearch) which encapsulates nearly all of the possible properties of an Asteroid. Additionally, because there may be reporting requirements in the future to track things like Mission success rate, minerals found, etc., the AsteroidSearch allows users to specify *ranges of criteria*, and find all Asteroids that have properties within the specified range (see the usage of the Apache Commons org.apache.commons.lang3.Range<T> object to assist with range-based search criteria). Further, I felt that the Search capability should not only be limited to Asteroids, but also to Missions and Spacecraft as well, and so this design includes rich object-based search criteria for all types. This extends the cscie97.asn3.ecommerce.collections.ContentSearch design that I did for Assignment 3.

Another slight change is that this design calls for enumeration types in several places, in each sub-system, and these enumeration types should be stored in their own database tables. Using enumerations allows for better code re-use, and can also support expanding the list of potential enumerations by simply adding new rows to the associated database table and re-starting the application (upon application start the enumeration classes should have the possible values read in from the database).



Perhaps a somewhat larger change is that this design calls for any sort of notes, observations, discoveries, or messages to be implemented as a List or Set; the original requirements only call for single values for things such as Asteroid “ExplorationStatus”, but such a design would be overwriting old discovery values when new ones are made. Rather this design seeks to create a rich archive of all discoveries made by each Spacecraft as dated records, to note the minerals found and in what quantities and when for each Asteroid, etc. There is some additional database complexity to be considered when implementing this design over the original requirements, but the ultimate result of implementing this should be a much more usable system that provides extremely valuable historical information to users.

Another smaller change is that Missions can support N-number of Spacecraft, rather than just one Spacecraft. For Missions to larger or more distant Asteroids, it may become too risky, impractical, or even impossible to send a viable exploratory or mining mission with only a single Spacecraft.

## Testing

Since the application is intended to be entirely database-backed, a testing library should consist of the following:

- A set of SQL scripts that populates all of the numeration type values, each in their own database table
- A set of SQL scripts that creates all of the base object records in the database (Asteroid, Mission, Spacecraft)
- A set of SQL scripts that creates all of the extended properties of those base object records (e.g., add ExplorationStatus for Asteroids, DiscoveryMessage records for Spacecraft, etc.)
- An object loader, that will iterate over all the tables in the database, load each record, and transform that record into an object that resides in memory

Further, following Test-Driven Development (TDD) principles, a testing library should also exercise all parts of the system that interact with the database, including creating new records, updating existing records, and deleting old records. Testing should include attempts to delete non-existent records, updating non-existent records, concurrency based testing, etc. Given the database-driven nature of the planned implementation, a CSV-file loading mechanism as was used for the prior assignments is not really appropriate here (either as a means of populating the data or as a way of testing the correctness of the overall system).

Additionally, this system has a user front-end in the Command & Control User Interface, which will require extensive testing to verify that the forms are properly sending data to the object model, and that the objects are properly persisting in the database.

Finally, as in Assignment 4, the AuthenticationServiceAPI should be tested to verify that users saved in the database are able to properly authenticate against the system, log in, log out, etc. Tests should be robust and include a variety of scenarios, including testing a

logged in user attempting to access restricted interface methods with an expired accessToken, etc.

## Risks

An admitted shortcoming in this design for the overall system is the message interchange between Spacecraft and the Mission Management System. The current design has Spacecraft calling a message on the IEarthBasedCommunicationLink when they make a new discovery of resources, water, or life, and then the EarthBasedCommunicationLink sending that message to the MissionManagementAPI to update the statuses of the associated Asteroid and Spacecraft. A much more robust and frankly better architecture would be to use an Enterprise Service Bus to enable message archiving, queueing, resiliency, tracking, and also two-way data interchange (for example, so that the Mission Management System could thus create and send messages to the Spacecraft). By using an ESB, Spacecraft could simply publish any new DiscoveryMessages to the service bus, and the bus would handle routing, brokering publish/subscribe relationships between objects/components, etc.

Another risk area is the database itself; as the system is designed to be database-driven, a database failure could be catastrophic for the overall system. Therefore, a system of multiple redundancies and failover plans should be in place (database replication, remote availability, etc.).