DEVCOM DAC-TR-2022-112
December 2022

The Soldier and Squad Trade Space Analysis Framework (SSTAF) Interactive Analyzer

by Ronald A. Bowers

DISCLAIMER

The findings in this report are not to be construed as an official Department of the Army position unless so specified by other official documentation.

WARNING

Information and data contained in this document are based on the input available at the time of preparation.

TRADE NAMES

The use of trade names in this report does not constitute an official endorsement or approval of the use of such commercial hardware or software. The report may not be cited for purposes of advertisement.

DEVCOM DAC-TR-2022-112
December 2022

The Soldier and Squad Trade Space Analysis Framework (SSTAF) Interactive Analyzer

by Ronald A Bowers

DEVCOM Analysis Center

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED (From - To)
December 2022	Technical Report	February–September 2022

4. TITLE AND SUBTITLE
The Soldier and Squad Trade Space Analysis Framework (SSTAF) Interactive Analyzer

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Ronald A. Bowers

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Director
DEVCOM Analysis Center
6896 Mauchly Street
Aberdeen Proving Ground, MD 21005

8. PERFORMING ORGANIZATION REPORT NUMBER

DEVCOM DAC-TR-2022-112

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT
DISTRIBUTION STATEMENT A. Approved for public release: distribution unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
(U) The U.S. Army is seeking to accelerate the development of systems to counter near-peer adversaries in a multidomain environment and plans to leverage modeling and simulation to help guide its research, development, engineering, and acquisition efforts. To provide the required capability of estimating Soldier performance in a combat environment, the U.S. Army Combat Capabilities Development Command (DEVCOM) Analysis Center, known as DAC, is developing the Soldier and Squad Trade Space Analysis Framework (SSTAF). SSTAF is a software infrastructure system for integrating multiple human performance and other models to provide a unified representation of Soldier state, capability, and behavior. SSTAF models the Soldier as a system, where the results of one model can affect the results of other models, and both the positive and negative effects of Soldier equipment can be captured. The SSTAF Interactive Analyzer is an enhancement to SSTAF that provides a self-contained analysis capability. The Interactive Analyzer enables evaluation of analysis scripts and supports integration of SSTAF capability into applications regardless of implementation language.

15. SUBJECT TERMS
Soldier performance, squad performance, Soldier lethality, human performance, modeling and simulation

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Ronald A. Bowers
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UU	829	19b. TELEPHONE NUMBER (include area code) (410) 278-3348

Table of Contents

Table of Contents

List of Figures

List of Code Excerpts

Acknowledgments

The U.S. Army Combat Capabilities Development Command (DEVCOM) Analysis Center, known as DAC, recognizes the following individuals for their contributions to this report:

John Gantert, DAC
Gregory Dietrich, DAC
Timothy Myers, DAC
Stephanie Snead, DAC

Executive Summary

This report documents the design and use of the Interactive Analyzer application for the Soldier and Squad Trade Space Analysis Framework (SSTAF). SSTAF is a software infrastructure system for integrating multiple human performance and other models to provide a unified representation of Soldier state, capability, and behavior. SSTAF models the Soldier as a system where the behavior of one model can affect the behavior of other models, and both the positive and negative effects of Soldier equipment are represented. The goal of SSTAF is to provide an architecture that enables the development of digital twins for specific Soldiers. The Interactive Analyzer provides a new ability to run SSTAF scripts and more easily integrate SSTAF capability into force-level models and other applications.

1. INTRODUCTION

This report documents the design and use of the Interactive Analyzer application for the Soldier and Squad Trade Space Analysis Framework (SSTAF). SSTAF is a software system for integrating multiple human performance and other models to provide a unified representation of Soldier state, capability, and behavior. SSTAF models the Soldier as a system, where the results of one model can affect the results of other models, and both the positive and negative effects of Soldier equipment can be captured. The goal of SSTAF is to provide an architecture that enables the development of *digital twins* for Soldiers. These digital twins can be used not only for material trade space analysis but also for interactive training and mission planning. A detailed description of the SSTAF architecture is provided in Bowers (2021).

1.1 Overview of SSTAF

An overview of the SSTAF concept is shown in Figure 1. The box at the bottom shows the SSTAF system. Multiple models that predict various aspects of the Soldier can be loaded into the framework. SSTAF models can depend on other models. SSTAF provides mechanisms to reconcile dependencies between models and enable models to update each other according to what has occurred in the simulation.

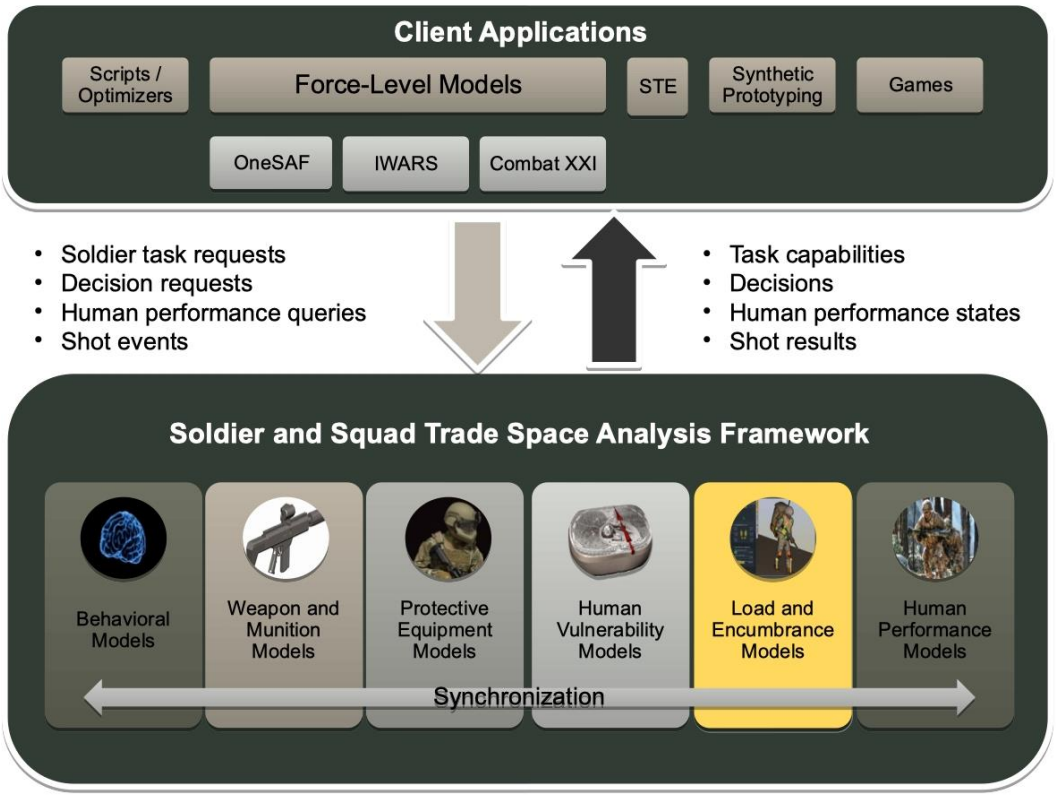

Figure 1. SSTAF concept

The upper box shows how SSTAF can be integrated into client applications to fulfill user-specific purposes. This aspect of SSTAF illustrates one of its primary merits. SSTAF provides model developers with a single target for integration. Rather than having to integrate models into multiple environments, models can be integrated into SSTAF and SSTAF provides the integration into higher-level constructs. This approach provides two significant benefits. First, it reduces development costs and repeated work since both models and client applications can program to stable SSTAF interfaces. Second, this approach helps ensure that human performance and behavior are represented consistently across different environments.

1.2 Motivation for the Interactive Analyzer

One limitation of the initial version of SSTAF was that it required client applications be written either in Java or in another Java virtual machine (JVM)-compatible language (e.g., Scala or Kotlin). This was necessary so that the SSTAF infrastructure, the SSTAF modules, and the client code could be loaded into a common JVM. The SSTAF Interactive Analyzer eliminates this requirement and facilitates the different application types shown in the upper box of Figure 1. It does so by encapsulating SSTAF capability into a standalone application. This application can be used interactively by other simulations by passing messages to the analyzer and receiving results. The Interactive Analyzer can also read the SSTAF commands' file and write results to another file, thereby providing the ability to execute analysis scripts.

There is an advantage to decoupling SSTAF from the client application even when the client application is JVM-based. Executing SSTAF in a standalone application enables the use of more advanced versions of Java within SSTAF. Previously, SSTAF was limited to using Java 11 as this is the version that is used by OneSAF. The current version of Java is Java 19, which has several features that could be very useful in SSTAF. One such feature is modernized foreign function and memory access. This feature will make it easier to interact with C and C++ libraries, such as the Operational Requirements-based Casualty Assessment (ORCA) model.

1.3 Purpose and Organization of the Report

This report serves multiple purposes. First, it documents the design of the SSTAF Interactive Analyzer to help those who might maintain and extend it. Second, it instructs developers on how to integrate the Interactive Analyzer into client applications. Third, it describes how to prepare input and use the Interactive Analyzer. Finally, it documents the source code for SSTAF in the appendices.

1.4 Conventions

Typography and grammar conventions are used to convey the category of an item when it is described in the report. The convention used is as follows:

- Java classes are depicted in a fixed-width font, for example, `Entity`.
- Class names are not pluralized. Phrases such as "`Entity` instances" are used instead of "`Entities`."
- Within the text, Java variables and JavaScript Object Notation (JSON) keywords are depicted in a fixed-width, italic font, for example, ***`mass_kg`***.
- Within JSON examples, keys and values are depicted in different colors, for example, **"mass_kg" : 3.14**.

2. DESIGN OF THE INTERACTIVE ANALYZER

As shown in Figure 2, the SSTAF Interactive Analyzer is a simple application that leverages the SSTAF architecture. It has only two responsibilities. The first is to instantiate a SSTAF **`Session`**. The second is to receive and process commands from the client application or input file using a “Read-Evaluate-Print Loop” (REPL).

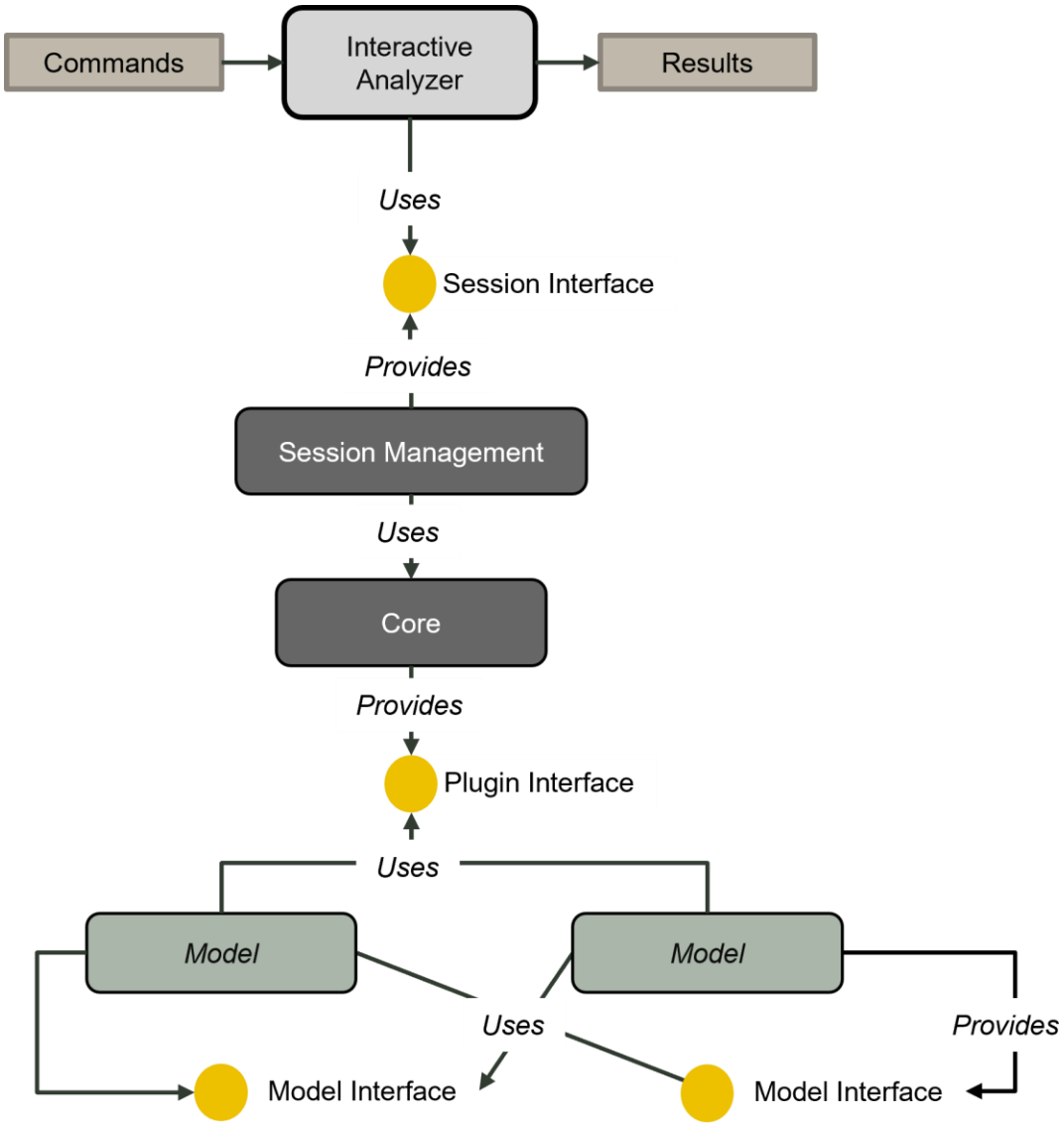

Figure 2. Architecture of SSTAF and the Interactive Analyzer

2.1 Instantiating the Session

The **Session** is instantiated by loading the input graph from one or more input files. The input graph consists of the definitions for all the **Entity** instances (i.e., instances of **Soldier** and Unit) that are to be represented by the system as well as the specification and configuration for all the models that are to be used with those instances. SSTAF input specifications are described in detail in Section 3.

As the input loader traverses the input graph, each **Entity** definition encountered is loaded and the instance is instantiated. The **Feature** implementations that have been declared to be used with it are loaded and any additional dependencies are resolved. Once this process is completed, the **Session** is ready to receive commands through its interface and the Interactive Analyzer enters the REPL.

If the analyzer has been configured to read from a file, the file must contain the series of commands for the analyzer to execute. The format for that file is described in Section 4.

2.2 Read-Evaluate-Print Loop (REPL)

The main component of the Interactive Analyzer is the REPL. An REPL is an architectural feature common to interactive command interpreters like the Interactive Analyzer. As the name implies, an REPL performs three actions. First, it reads input from the user or another input source. Next, it evaluates the input that was read. Finally, it prints the result of the evaluation back to the user or to another output destination. The execution loop for the Interactive Analyzer is shown in Code Excerpt 1.

The first action performed by the REPL is to read the input as shown at Line 1. The ***inputSupplier*** delegate reads from an **InputStream** attached to standard input (stdin) and provides the input as a **String** to the analyzer. After some checks, the input is passed to the ***deserializer*** delegate at Line 2. The ***deserializer*** uses the Jackson JSON library to parse the input string and generate a Java object that is a subclass of **BaseAnalyzerCommand**. Next, at Line 3, the class of the **BaseAnalyzerCommand** is used to look up the appropriate strategy to evaluate the command that was provided. If a strategy is found in the ***processingStrategyMap***, it is invoked, and the result captured (Line 4). To print the result, the REPL starts by serializing the result from a Java object into a JSON object written into a **String**. The ***serializer*** delegate at Line 5 performs this action. Finally, at Line 6, the output is written using the ***outputConsumer*** delegate.

The read and print phases of the REPL are split into two actions. Each action is performed by a delegate that is assigned when the analyzer is instantiated. The Interactive Analyzer uses a factory idiom (Bloch, 2008) to build instances of the REPL

class. The factory creates each of the delegate objects and then provides them to the REPL constructor. Currently there is only one factory. This factory configures the REPL to read JSON objects from standard input (*stdin*) and write JSON objects to standard output (*stdout*). Future versions could add additional communication methods and encodings but there is no requirement for those capabilities at this time.

Code Excerpt 1. The Read-Evaluate-Print Loop

```
while (keepRunning.get()) {
    try {
        long startTime = System.currentTimeMillis();
        String input = inputSupplier.get();                        // 1
        logger.debug("Got {}", input);
        if (input != null && input.length() > 0) {
            BaseAnalyzerCommand command =
                deserializer.apply(input);                         // 2
            ProcessingStrategy strategy =
                processingStrategyMap.get(command.getClass());    // 3
            BaseAnalyzerResult result;

            if (strategy == null) {
                throw new SSTAFException(
                    "Unknown Analyzer Command");
            } else {
                result = strategy.process(command);                // 4
            }

            taskCounter.incrementAndGet();
            result.setId(command.getId());
            result.setProcessingTime_ms(
                System.currentTimeMillis() - startTime);
            String output = serializer.apply(result);             // 5
            outputConsumer.accept(output);                         // 6
            logger.info("Sending {}", output);
        } else if (input != null) {
            logger.info("Received 0-length command");
        } else {
            logger.info("Received null command");
        }
    } catch (Exception e) {
        e.printStackTrace();
        rv = -1;
        break;
    }
}
```

Currently there are three strategies for processing commands from the client. The first processes the command to retrieve the table of entities in the session. The second dispatches one or more messages to the entities in the session. The third shuts down the session. The messages associated with these strategies and their responses are documented in Section 4.

Those readers familiar with Java's input/output library might notice that it would be easy to have the analyzer use a network socket for communication. Doing so would enable additional capabilities such as hosting the analyzer on a separate computer from the client application. Unfortunately, enabling network communication in an application activates a multiplicity of software security requirements that make implementation much more complex than simply adding delegates for the socket streams and a new factory to assemble the REPL. A future version of the Interactive Analyzer might implement network communication, but it is not required at this time.

3. SSTAF ENTITY INPUT DEFINITION

SSTAF input is expressed as a directed acyclic graph specified using JSON objects and arrays. The graph defines the **`Entity`** instances that are used in the analysis, the **`Feature`** implementations (i.e., models) that are attached to each, and the **`Configuration`** objects specific to those **`Feature`** implementations. In this section, the structure of SSTAF input is described. First, a series of overarching topics is discussed, then the input graph is described in detail.

SSTAF input can be complex and difficult to assemble. Currently there are no tools available to simplify the process other than an editor that understands JSON syntax. Using a JSON editor can reduce simple errors such as mismatched quotes and braces, but it cannot validate the objects defined in the JSON file. The best way to debug SSTAF input is to start small and gradually increase complexity. For example, the input for individual **`Soldier`** instances can be configured and then grouped into a **`Unit`** after they are verified.

3.1 Object Class Specification

Originally SSTAF used custom JSON parsers to read its input. This required significant amounts of code—both to read the input and to properly handle implementing the necessary builders for class hierarchies. SSTAF now uses the *Jackson* library to read and write JSON-encoded information. The transition to *Jackson* greatly reduced the amount of code in the system but introduced a complication. It is now necessary to specify the type of each object in its JSON specification. An example is shown in Code Excerpt 2.

Code Excerpt 2. Class Specification in JSON Object

```
{
    "class":"object.class.name",
    // Additional class-specific fields
}
```

3.2 File References

One enhancement to JSON processing added to SSTAF is the ability to use file references. A file reference can be used anywhere a JSON object or JSON array is used in the input. File references simplify input development by making sections of the input smaller and easier to manage. They also enable the reuse of sections of the input graph.

A file reference is created by providing a string that can be interpreted as a file path and that ends in ".json" as the value for any key. When the SSTAF JSON loader encounters a file reference, it jumps to the referred file, processes it, and substitutes the processed JSON string into the referring JSON input. This process can recurse, thus, references can contain references. All references are resolved and the entire input graph is converted into a single JSON string before being deserialized.

File references can be either absolute or relative. If the reference is relative, the location is relative to the location of the file from which the referring object was loaded. The default parent directory is defined by the ***user.dir*** JVM property. Consequently, if the referring object was read from standard input, relative references will be resolved relative to the user's home directory.

The SSTAF input loader detects recursive references to the same file. This prevents cycles in the input and reduces the likelihood of out-of-memory errors. The same file reference can be used in different branches of the graph.

3.3 Common Input for All Entity Types

As described in Bowers (2021), the software classes used to represent humans, Soldiers, and units are all subclasses of SSTAF's **Entity** base class. The **Entity** class contains the infrastructure for accepting **Feature** implementations as well as the mechanism for responding to requests from other entities. All **Entity** implementations extend a common base type and thus also share several input elements. The common elements are shown in Code Excerpt 3.

At Line 3 is the list of **Feature** implementations that are to be added to the **Entity**. A **Feature** is a plugin that adds some capability to the **Entity** (Bowers, 2021). Each entry in the ***features*** list is an instance of **FeatureSpecification**. The contents of a **FeatureSpecification** are ***featureName***, ***majorVersion***, ***minorVersion***, and ***requireExact***. The ***featureName*** field specifies the human-friendly name of the feature. Examples include "ANSUR Anthropometry" and "Telemetry." The ***majorVersion***, ***minorVersion***, and ***requireExact*** fields specify the version constraints on the requested **Feature**. If ***requireExact*** is false, any **Feature** with the specified name with a version greater than or equal to the specified version will be considered a match. If the ***requireExact*** is true, only a version that matches the ***majorVersion*** and ***minorVersion*** will be loaded. Since the feature list can only contain instances of **FeatureSpecification**, it is not necessary to specify the class type in the **FeatureSpecification** entries.

The ***majorVersion***, ***minorVersion,*** and ***requireExact*** fields are optional. The default value for both ***majorVersion*** and ***minorVersion*** is 0, and the default value for ***requireExact*** is false. Thus, it is possible to specify only the ***majorVersion*** and get any version that meets or exceeds the specified ***majorVersion*** value. Similarly, if all three fields are omitted, any **Feature** implementation that matches the specified name is eligible for use. If multiple matches are found, the implementation with the highest version number will be selected for use.

The next section of input starts at Line 4. This section specifies the **configuration** to be provided to the **Feature** instances. Each key in the ***configurations*** section (see Line 5) corresponds to a **Feature** that is loaded into the **Entity**. Note that this includes both the Feature instances loaded directly using the features list as well as indirectly through **@Requires** dependencies within those **Feature** instances. An example of this situation is that the Santos human dynamics model uses the ANSUR anthropometry service to configure its avatar. When using Santos, the analyst will want to provide a configuration to “ANSUR Anthropometry.”

Code Excerpt 3. Common Input Fields for Entities

```
{
  "class": "mil.sstaf.….",                                  // 1
  "randomSeed": 3                                           // 2
  "features": [                                             // 3
    {
      "featureName" : "Name of Feature",
      "majorVersion" : 1,
      "minorVersion" : 0,
      "requireExact" : true | false
    }
  ],
  "configurations": {                                       // 4
    "Name of Feature" : {                                   // 5
      "class" : "configuration.class.name",                 // 6
      "key1" : "value1",
      "keyN" : "valueN"
    }
  },
  "moduleLayerDefinition" : {                               // 7
    "modules": [                                            // 8
      "module1",
      "module2"
    ],
    "modulePaths": [                                        // 9
      "path/to/modules1",
      "path/to/modules2"
    ]
  }
}
```

As shown at Line 6, the value for each key in the **`configuration`** is an object that specifies the options for the **`Feature`**. Each **`Feature`** has its own **`configuration`** class, which is a subclass of the base **`FeatureConfiguration`** class. The base **`FeatureConfiguration`** class includes one field, ***`seed`***, that is used to initialize any random number generator in the **`Feature`**. Setting the ***`seed`*** field in the configuration is optional. If it is not set, the random number sequence in the **`Feature`** will be initialized

from the random number generator in the containing **Entity**. The specific configuration options for each **Feature** will be described in the sections documenting that **Feature**.

A new capability added to SSTAF since the publication of Bowers (2021) is the introduction of support for Java module layers. Module layers enable the Java runtime to compartmentalize dynamically loaded classes and constrain their visibility. They also provide support for resolving dependencies outside of the paths specified in the ***modulepath*** parameter declared when the application was started. Each **Entity** can declare its own instance of a **ModuleLayer**. This enables each **Entity** to specify which modules include the **Feature** implementations that will be loaded into the **Entity**. Specifying a **ModuleLayer** is optional. If not specified, **Feature** implementations will be resolved according to the runtime ***modulepath***.

Configuration of the **ModuleLayer** begins with the ***moduleLayerDefinition*** keyword at Line 7. At Line 8, the ***modules*** keyword specifies a list of one or more modules that are to be loaded by the **Entity**. At Line 9, the ***modulePaths*** keyword specifies a list of one or more directories in which the modules can be found. It is not necessary to align the module with the paths. The system will search all paths for all modules. Note that both absolute and relative paths are supported. If the path is relative, it will be resolved relative to the current input file. This facilitates packaging **Feature** modules with the entities that use them.

One additional property of module layers is that they are hierarchical. For example, if the configuration for a **Unit** contains a **ModuleLayerDefinition**, that definition will be inherited by all **Entity** instances loaded under that **Unit**, including both **Soldier** instances and other **Unit** instances. Those child **Entity** instances can add additional layers through their configurations, but they cannot eliminate, or modify, the parent layer.

3.4 Input Graph Structure

The root of the SSTAF input graph is the configuration for an instance of the **EntityController** class. The **EntityController** has two functions. First, it is the container that holds all the **Unit** and **Soldier** instances in the system. Second, it manages communication to and from those entities and between those entities and the client application. Under realistic use cases, this configuration will be in a separate JSON file and file references will be used to load the various entities that are used. An example of an **EntityController** configuration is shown in Code Excerpt 4.

The **EntityController** is also an **Entity**. Because it is an **Entity**, **Feature** implementations can be added to it. However, no **Feature** types have been developed

for use in the **EntityController** at this time. This lack is reflected in the empty ***features*** and ***configurations*** blocks at Lines 3 and 4. As explained in Section 3.3, if a ***moduleLayerDefinition*** entry is included in the **EntityController**, that layer becomes the parent layer for all instances of **Entity** that are loaded into the system.

Code Excerpt 4. Sample EntityController Configuration File

```
{
  "class": "mil.sstaf.session.control.EntityController",
  "randomSeed": 3
  "entities": {                                                    // 1
    "BLUE": [                                                      // 2
      "UnitConfiguration/BluePlatoon.json",
    ],
    "RED": [
      "UnitConfiguration/RedPlatoon1.json",
      "UnitConfiguration/RedPlatoon2.json"
    ],
    "GRAY": [],
    "GREEN": []
  },
  "features": [],                                                  // 3
  "configurations": {}                                             // 4
}
```

The ***entities*** key at Line 1 indicates the object that defines the entities that will be loaded into the analysis. Within the entities object are a set of keys that indicate the **Force** to which the entities belong, and a list of the **Entity** instances associated with that force (Line 2). A **Soldier** or **Unit** instance can be associated with one of four forces: BLUE, RED, GRAY, or GREEN. There is no analytic meaning to the forces or their labels. It is simply a way to divide the entities into groups that are not within a common hierarchy. Each force may have any number of entities in it. It is not necessary to include empty forces in the specification.

3.4.1 Unit Definition

In SSTAF, a unit is an organizational construct that can contain one or more Soldiers directly and can also include one or more subunits. It is represented by the **Unit** class. There is no limit to the number of Soldiers who are members of the unit or the number of subunits. There is also no limit to the number of layers in the hierarchy. An example of a configuration file for an instance of **Unit** is shown in Code Excerpt 5.

Code Excerpt 5. Sample Unit Configuration

```
{
  "class" : "mil.sstaf.core.entity.Unit",
  "name": "Test Platoon",                                          // 1
  "features": [],                                                  // 2
  "configurations": {},                                            // 3
  "soldiers": [                                                    // 4
    {
      "position": "Leader",                                        // 5
      "soldier": "../SoldierConfiguration/TestSoldier1.json"       // 6
    },
    {
      "position": "Medic",
      "soldier": "../SoldierConfiguration/Medic.json"
    }
  ],
  "subUnits": [                                                    // 7
    {
      "label": "Squad A",                                          // 8
      "unit": "Squad.json"                                         // 9
    },
    {
      "label": "Squad B",
      "unit": "Squad.json"
    },
    {
      "label": "Squad C",
      "unit": "Squad.json"
    }
  ]
}
```

The name of the top-level unit is set at Line 1. The value of the name may be anything. When and how the name of a unit is used will be described in the discussion of paths in Section 4.2.

Since a **`Unit`** is an **`Entity`**, **`Feature`** implementations can be added to it and configurations supplied to the features. Currently there are no **`Feature`** implementations that operate on units. Therefore, Lines 2 and 3 can be omitted. It is also possible to add

a ***moduleLayerDefinition*** specification. If done, that module layer will be the parent layer for the **Unit** and **Soldier** instances within this unit's hierarchy.

The specification for the Soldiers that are organic to the unit starts at Line 4. The value for the ***soldiers*** keyword is a list of **Soldier** entries. Each entry has mandatory ***position*** (Line 5) and ***Soldier*** fields (Line 6). The ***position*** field defines the role of the Soldier within the unit, for example, "Leader," "Medic," or "Chaplain." Each ***position*** value must be unique within the unit. The ***Soldier*** field specifies the **Entity** configuration for the Soldier. Although it is possible to embed a **Soldier** definition within a unit definition, it is much more practical to use a file reference as was done in the example.

The specification for subunits begins at Line 7 with the ***subUnits*** keyword. The value of the ***subUnits*** keyword is a list of subunit entries. Each entry has mandatory ***label*** (Line 8) and ***unit*** (Line 9) fields. Like the ***position*** field described previously, the ***label*** field is a unique identifier for the subunit within the unit. The ***unit*** keyword specifies the **Entity** configuration for the subunit. As is the case with Soldiers, it is more practical to use a file reference rather than embed a unit definition within a unit.

As stated previously, the SSTAF JSON loader provides protection against circular references. Loading will fail if the system encounters a reference to a unit that it is currently traversing. However, note that it is possible to use the same definition for both Soldiers and units in multiple places so long as circular references are not created. By reusing Soldier and unit definitions it is possible to create large input graphs with a minimal amount of input.

3.4.2 Soldier Definition

The Soldier is the primary construct in SSTAF. Currently, all models that are available are related to predicting the capability, state, and behavior of individual Soldiers. That said, the input for an instance of **Soldier** is trivially different from that common to all **Entity** types. An example of Soldier input is shown in Code Excerpt 6. This example makes use of the default values in the **Feature** specifications to reduce unnecessary values in the file.

Code Excerpt 6. Sample Soldier Configuration

```
{
  "class" : "mil.sstaf.core.entity.Soldier",
  "name": "Alvin York",                                          // 1
  "rank": "E5",                                                  // 2
  "features": [                                                  // 3
    {
      "featureName": "Santos Dynamic",
      "majorVersion": 1,
      "minorVersion": 0,
      "requireExact": "false"
    }, {
      "featureName": "Dynamic Aim"
    }, {
      "featureName": "Kit Manager"
    }, {
      "featureName": "Telemetry Agent"
    }
  ],
  "configurations": {                                            // 4
    "ANSUR Anthropometry": {                                     // 5
      "constraints": [
        {
          "propertyName": "PrimaryMOS",
          "matches": "11B"
        }, {
          "propertyName" : "stature",
          "lowerBound" : 1700
        }
      ]
    },
    "Santos Dynamics": { },
    "Telemetry Agent": "../common/telemetryConfig.json",
    "Kit Manager": "../common/StandardKit.json",
    "Dynamic Aim": "../common/dynamicAimConfig.json"
  }
}
```

The specification for a Soldier has two optional fields in addition to the common fields for entities. They are ***name*** and ***rank.*** The value for a name may be any alphanumeric string. The value for rank is specified using the values E1 through E9 and O1 through O10. Any other value will cause an exception to be thrown and loading to fail. If a value is not provided, rank will default to E1. Currently, the rank value is not used by any **Feature**.

Starting at Line 3 is the features' specification block. This example adds four **Feature** implementations to the Soldier. They are:

1. "Santos Dynamics" – Predicts the motion of a Soldier using the Santos human dynamics model from the University of Iowa.
2. "Dynamic Aim" – Computes Soldier aim error based on the physical state of the Soldier. This is an experimental demonstration model for which there has been no validation effort.
3. "Kit Manager" – Manages the weapons and equipment carried by the Soldier and provides information such as weight.
4. "Telemetry Agent" – Records internal state from the models to a file.

The specification for the configurations for each of the features starts at Line 4. The interesting entry is the configuration for "ANSUR Anthropometry" at Line 5. This is an example of a feature being loaded through a @Requires dependency. Although support for ANSUR anthropometry was not requested in the Soldier configuration, it is required by the Santos feature, so it was loaded and is configurable. The documentation for a **Feature** should list its dependencies. If a **Feature** requires configuration but one is not provided, an exception will be thrown, and loading will fail.

4. COMMAND SYSTEM

The Interactive Analyzer is controlled using command messages. Like SSTAF input, the messages are expressed as JSON text. The responses from the analyzer are also in the form of message objects serialized as JSON text.

Because of the need to route messages through the layers of SSTAF, the command messages have a nested structure. The innermost elements are the commands that are sent to the **`Feature`** implementations within the **`Entity`** instances. These **`HandlerContent`** messages are described in Section 4.1. The next higher layer are objects that enable the routing of the messages through the SSTAF **`Session`** to the **`Entity`** instances. These session messages are described in Section 4.3. The outermost layer comprises the messages that go between the client or input file and the Interactive Analyzer itself. Analyzer messages are described in Section 4.4.

Like the command messages, the response messages are also nested. The innermost layer is also composed of **`HandlerContent`** objects. The **`HandlerContent`** objects are wrapped in session result objects, which are then wrapped in analyzer result objects. The result objects will be discussed with their command object peers.

4.1 HandlerContent Objects

In SSTAF, a **`Handler`** is a type of **`Feature`** that can react to a message. Messages that are consumed or emitted from a **`Handler`** are subclasses of the **`HandlerContent`** class. The messages that can be accepted by an **`Entity`** are defined by the **`Handlers`** that have been loaded into it. Consequently, the messages available to interact with the Interactive Analyzer depend on the configuration of the entities and the hierarchy as defined in the input. Because **`HandlerContent`** classes are defined within each **`Handler`** and the **`HandlerContent`** class itself has no fields, there is no common JSON structure other than the requirement to specify the class, as was shown in Code Excerpt 2.

The output produced by a **`Handler`** is also an instance of **`HandlerContent`**. This enables it to be routed to another handler or back to the client.

The establishment of the **`HandlerContent`** superclass is a change from what is documented in Bowers (2021). Previously, messages to **`Handler`** implementations could be of any type. However, the change from handwritten JSON parsers to the Jackson JSON library required the establishment of a formal class hierarchy for the **`Handler`** messages.

4.2 Paths

Messages are routed to entities using a path. The path expresses the identity of the **Entity** within the hierarchy of forces, units, and Soldiers. The path is a formatted **String** and is intended to be human-readable.

An important rule when composing a path is that if an **Entity** is a member of a **Unit**, the path uses the identifier associated with that **Entity** in the containing **Unit**. To clarify, if addressing a **Soldier** within a Unit, one uses the Soldier's ***position*** as the identifier for the Soldier in the path. Similarly, if addressing a unit within another unit, the subunit's ***label*** is used. If the entity is not the member of a unit, then its ***name,*** as specified in the input, is used in the path.

The path starts with the name of the containing **Force** and includes the unit hierarchy down to the specific target entity. The general form of a path is:

Force:Unit1:Unit2:…:UnitN:receivingEntity

The value for ***Force*** is either ***RED, BLUE, GREEN, or GRAY*** and corresponds to the assignment provided in the input. The default value for ***Force*** is ***BLUE***, so if one is addressing a ***BLUE*** entity, the ***Force*** field can be omitted. The chain of "unit" fields corresponds to the unit hierarchy that contains the receiving entity.

4.3 Session Messages

As stated previously, **HandlerContent** objects are wrapped in session messages. Session messages are responsible for getting a message to the desired entity. The class hierarchy for session messages is shown in Figure 3. At the root of the session message hierarchy is the **BaseSessionCommand**. Although this is a not an abstract class, it is never used directly. The **BaseSessionCommand** includes one instance field, ***id***, which contains the sequence number for the command.

The **Command** class extends **BaseSessionCommand**. It includes three fields: ***recipientPath***, ***content***, and ***handle***. The ***recipientPath*** is the path string that specifies the ***Entity*** to which the message is being sent. The ***content*** field contains the **HandlerContent** object that will be delivered to the **Handler** within the **Entity**. The ***handle*** field is the SSTAF's internal identifier for the receiving **Entity**. When a Session message is sent into the **EntityController**, the path is deciphered to determine the appropriate destination and retrieve the **Handle** for that **Entity**. A new internal message object is then created that uses the **Handle** to indicate the recipient. The **HandlerContent** object is attached to this new message and the new message is dispatched to the receiving **Entity**. A sample of the JSON for a **Command** object is shown in Code Excerpt 7.

The **Event** class extends the **Command** class. An **Event** is a **Command** that is to be executed at some specific simulation time rather than immediately. **Event** objects are dispatched to the entity when the simulation time reaches the event time. A sample of the JSON for an **Event** object is shown in Code Excerpt 8.

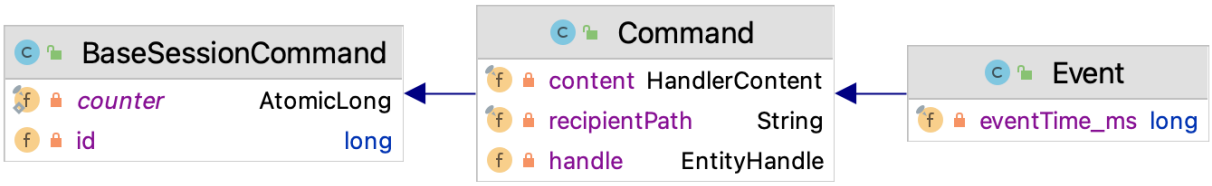

Figure 3. Class hierarchy for session commands

Code Excerpt 7. Sample JSON for Session Command

```
{
    "class" : "mil.sstaf.session.messages.Command",
    "recipientPath" : "Force:Unit1:Unit1.1:Unit1.1.1:Entity",
    "content" : {
      "class" : "mil.devcom_sc.ansur.messages.GetValue",
      …
    }
}
```

Code Excerpt 8. Sample JSON for Session Event

```
{
    "class" : "mil.sstaf.session.messages.Event",
    "recipientPath" : "Force:Unit1:Unit1.1:Unit1.1.1:Entity",
    "eventTime_ms" : 3600000,
    "content" : {
      "class" : "mil.devcom_dac.equipment.messages.AddItem",
      …
    }
}
```

The class hierarchy for the session responses is shown in Figure 4. The base class is **BaseSessionResult**. It contains a single field, ***id***, which corresponds to the identification number of the command for which it is the product. The primary implementation of **BaseSessionResult** is **SessionTickResult**. **SessionTickResult** contains two additional

fields: ***messagesToClient*** and ***nextEventTime_ms***. The first contains all of the outbound messages from the session. The second contains the simulation time for the next scheduled event. This is provided so the client application can advance the clock to that point if desired, but it is not required to do so. Clients can advance the clock at any interval.

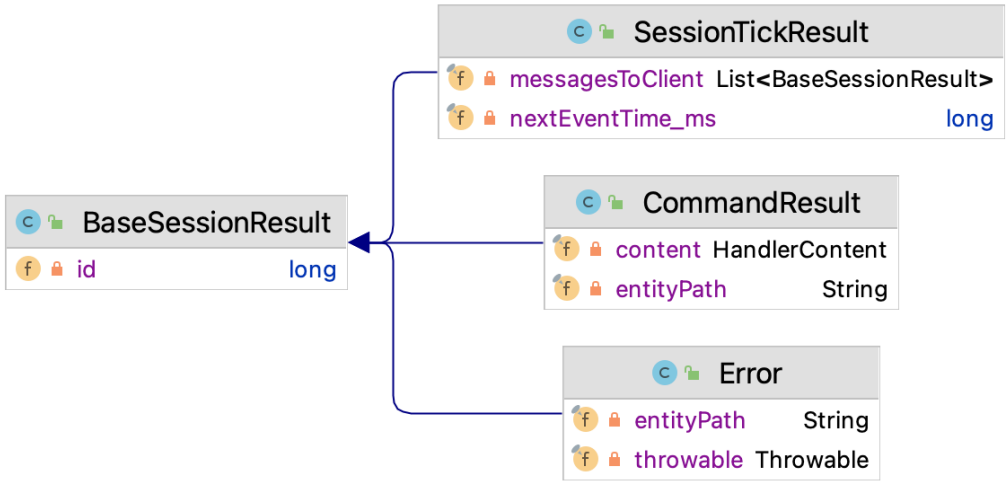

Figure 4. Class hierarchy for session responses

The contents of the ***messagesToClient*** list are either **`CommandResult`** or **`Error`** objects. A **`CommandResult`** contains the path to the source **`Entity`** in the ***`entityPath`*** field. The outbound message itself is in the ***`content`*** field. If an exception occurs while processing a command, the exception is packaged in an **`Error`** object. A sample of JSON for a session response is shown in Code Excerpt 9. The serialized exception specified by the **`throwable`** key is generated automatically by the *Jackson* library. It will contain the exception message, the stack trace, and an underlying cause exception if one was set. The stack trace can be large and will be of little value to the end user, so it would be more reasonable to log the JSON text and present only the **`message`** field from the **`Throwable`** to the user.

Code Excerpt 9. Sample JSON for Session Result

```
{
  "class" : "mil.sstaf.session.messages.SessionTickResult",
  "nextEventTime_ms" : 8675309,
  "messagesToClient" : [
    {
      "class" : "mil.sstaf.session.messages.CommandResult",
      "entityPath" : "Force:Unit2":Unit2.1:Entity",
      "content" : {
        // Handler specific information
      },{
      "class" : "mil.sstaf.session.messages.Error",
      "entityPath" : "Force:Unit2":Unit2.2:Entity2",
      "throwable" : {
         // Serialized exeption
      }
      …
    }
  ]
}
```

4.4 Analyzer Messages

The outermost messages used to control the Interactive Analyzer are the analyzer messages. The class hierarchy for the analyzer messages used is shown in Figure 5. The class hierarchy for the responses is shown in Figure 6. There is not a one-to-one correspondence because both the **`Tick`** and **`CommandList`** commands produce a **`TickResult`** response, and the **`Exit`** command does not have a response. The use of each of the commands, their JSON format, and the JSON format of their responses will be discussed in the following.

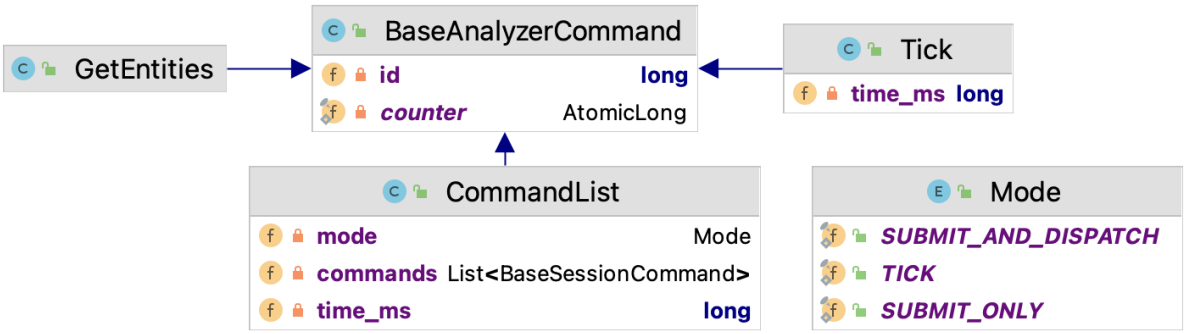

Figure 5. Class hierarchy for analyzer commands

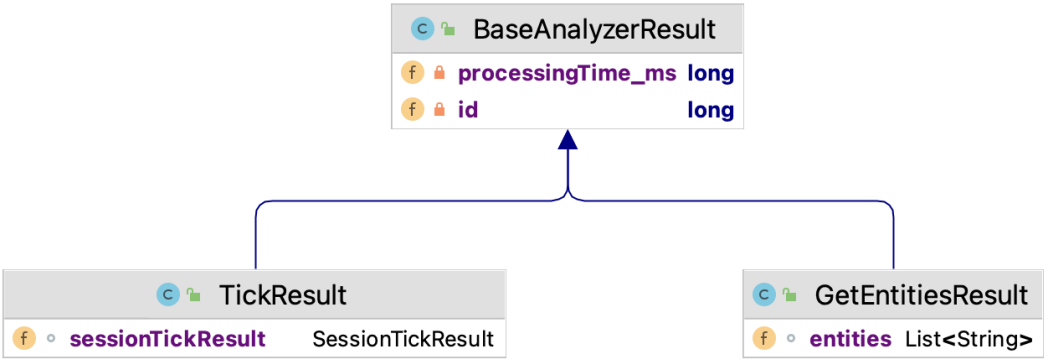

Figure 6. Class hierarchy for analyzer results

4.4.1 BaseAnalyzerCommand and BaseAnalyzerResult

The root of the analyzer command class hierarchy is the **`BaseAnalyzerCommand`**. Although this class is not abstract, it should never be used directly by a client application or in a script. The one feature of the **`BaseAnalyzerCommand`** to note is the ***`id`*** counter. This counter is set automatically by the Interactive Analyzer when the command is received and deserialized.

The root of the result hierarchy is the **`BaseAnalyzerResult,`** which has two fields. The first is the ***`id`*** number that was set in the corresponding command. The second, ***`processingTime_ms`***, is the time required to process the command in milliseconds.

4.4.2 GetEntities and GetEntitiesResult

The **`GetEntities`** command queries the analyzer for the list of entities that were loaded into the system. Its JSON form is shown in Code Excerpt 10. There are no arguments to the **`GetEntities`** command. The **`GetEntitiesResult`** contains the list of entities. Each

entry in the list is the full path to an entity. The JSON form for the result is shown in Code Excerpt 11.

Code Excerpt 10. JSON Specification for GetEntities Command

```
{
    "class" : "mil.sstaf.analyzer.messages.GetEntities"
}
```

Code Excerpt 11. JSON Specification for GetEntitiesResult

```
{
    "class" : "mil.sstaf.analyzer.messages.GetEntitiesResult",
    "entities" : [
        "Force:Unit1:Unit1.1:…:Entity",
        …
    ]
}
```

4.4.3 CommandList, Tick, and TickResult

The **`CommandList`** message provides the container for sending commands to individual entities. The commands are provided in the ***`commands`*** list, with each element in the list also being an instance of **`BaseSessionCommand`**.

The behavior of the **`CommandList`** command is controlled by the ***`mode`*** parameter and can be set to one of three values:

1. ***`SUBMIT_ONLY`*** instructs the analyzer to simply enqueue the commands for future delivery to the entities. The commands are not processed.
2. ***`SUBMIT_AND_DISPATCH`*** instructs the analyzer to send the commands to the entities and process them without advancing the simulation clock. This mode is most useful for "getter" actions that retrieve values from the entities.
3. ***`TICK`*** instructs the analyzer to send the commands to the entities and to process them by advancing the clock forward to the time specified by the time_ms parameter. Any additional actions associated with advancing the clock, such as those implemented by Agents, will be performed.

The **`Tick`** command does not provide any messages to the entities. It simply instructs the analyzer to advance the clock to the specified time and perform any actions associated with ticking the clock. Sample JSON for the **`CommandList`** is shown in Code Excerpt 12. Sample JSON for the **`Tick`** command is shown in Code Excerpt 13.

Code Excerpt 12. Sample JSON for CommandList Message

```
{
    "class" : "mil.sstaf.analyzer.messages.CommandList",
    "commands" : [
        {
            "class" : "mil.sstaf.session.messages.Command",
            "recipientPath" : "FORCE:Unit1:Unit1.1:…:Entity",
            "content" : {…}
        },
        …
    ]
    "mode" : "TICK", // or SUBMIT_ONLY or SUBMIT_AND_DISPATCH
    "time_ms" : 31415
}
```

Code Excerpt 13. Sample JSON for Tick Message

```
{
    "class" : "mil.sstaf.analyzer.messages.Tick",
    "time_ms" : 31415
}
```

The results for both the **`CommandList`** and the **`Tick`** command are packaged in a **`TickResult`** message. The TickResult contains a **`SessionTickResult`** that contains the list of messages from the entities to the client as well as the time of the next scheduled event. A sample of the JSON for a **`TickResult`** is shown in Code Excerpt 14.

It should be obvious at this point that the message structure for SSTAF is complicated and difficult to manage by hand. Therefore, it is recommended that a JSON library be used to generate the commands and parse the response messages. This is easiest for Java-based applications since the SSTAF message classes can be used directly. Eventually SSTAF might be enhanced with message classes for other languages, but this improvement is not scheduled.

Code Excerpt 14. Sample JSON for TickResult

```
{
  "class" : "mil.sstaf.analyzer.messages.TickResult",
  "sessionTickResult" : {
      "class" : "mil.sstaf.session.messages.TickResult",
      "nextEventTime" : 271828,
      "messagesToClient" : [
      {
        "class" : "mil.sstaf.session.messages.CommandResult",
        …
      },
      {
        "class" : "mil.sstaf.session.messages.Error",
        …
      },
      …
    ]
  }
}
```

4.5 Command Script

A command script for the Interactive Analyzer consists of a series of analyzer commands. Because these commands are separate objects, a script file is not a valid JSON file.

Commands can be split over multiple lines when they are provided to the analyzer either from a file or from a client application. The message reader in the REPL reads each line of input and determines if a complete JSON expression has been provided. Completeness is determined by matching opening and closing braces (i.e., { and }) and square brackets (i.e., [and]). The system uses a simple count of the tokens to check if the expression is complete, but it does not check if the JSON is valid. The validity of the input is evaluated by the *Jackson* library when the JSON text is deserialized. Individual commands must be separated by a new line.

5. USING THE ANALYZER

There are two ways to use the Interactive Analyzer. The first is to have the analyzer read commands from an input file and write results to an output file. The second is to use the analyzer as a helper application for a client application. In both cases the Interactive Analyzer must be started, either from the command line or from within the client application.

5.1 Starting the Interactive Analyzer

Because the Interactive Analyzer is a Java application, it requires some configuration to start it. The start command is shown in Code Excerpt 15. The value of **`${MODULE_PATH}`** is a colon-delimited list of the directories in which the SSTAF core, session, and analyzer modules are located. Paths to additional modules used in the analysis can be included here or in the ***`moduleLayerDefinition`*** blocks in the input as was described earlier. The ***`entityConfigFile.json`*** is the file that contains the root of the input graph as described in Section 3.4.

Code Excerpt 15. Start Command for Interactive Analyzer

```
java --module-path=${MODULE_PATH} --module mil.sstaf.analyzer/mil.sstaf.analy
zer.Main entityConfigFile.json
```

5.2 Using the Analyzer with Files

To use the analyzer with file-based input and output, one simply starts the analyzer as shown previously and provides the input and output files using input/output redirection.

5.3 Using the Analyzer from a Client Application

The mechanism for using the Interactive Analyzer from a client application will depend on the language in which the client application is written. For Java clients, the most straightforward approach is to use the **`ProcessBuilder`** class. Once the external process has been started through the **`ProcessBuilder`**, it can be accessed through the input and output streams available from the **`Process`** instance. This mechanism is best seen in the actual SSTAF source, which is included in Appendix A. The ProcessBuilder mechanism is shown in **`SSTAFAnalyzerIntegrationTest`** on page 50.

5.4 Configuration

SSTAF provides an application-level configuration mechanism that enables SSTAF-based applications to set properties. The primary use of the configuration file is to

establish a module layer immediately above the root layer established by the command-line arguments to the JVM. The location of the configuration file can be defined explicitly by providing the ***mil.sstaf.configuration*** property to the JVM, or by setting the **SSTAF_CONFIGURATION** environment variable in the command shell. The default location for the configuration file is ***$HOME/SSTAFData/config.json***.

Please note that if a configuration is provided, or if the ***SSTAFData/config.json*** file exists, the configuration class used in the file must be in the module set defined on the command line when the JVM is started. Otherwise, a ClassNotFoundException will be thrown.

6. FEATURES, COMMANDS, AND RESULTS

Several plugin implementations of the **`Feature`**, **`Handler`**, and **`Agent`** interfaces have been developed for SSTAF, and more are planned. This section documents the use of some of these features. For each feature, a description of its capabilities is provided. If the feature accepts configuration, the configuration options are described. If the feature accepts commands from a client, the commands and their responses are described.

The SSTAF Feature implementations documented in this report are those that are suitable for unrestricted release. Features that implement restricted capabilities will be documented separately with appropriate distribution limits. Although only a subset of SSTAF features are documented here and included with the source code in the appendices, these examples should be sufficient to understand the mechanics of SSTAF and prepare the user for more complex simulations.

6.1 ANSUR Handler

The ANSUR **`Handler`** provides the ability to select a subject entry from the ANSUR II data set to use for as the anthropometric data for a Soldier. The ANSUR **`Handler`** enables selection of the subject data either randomly or explicitly by specifying the subject's identification number. Random selection can be controlled by placing constraints on any of the values in the data set. If constraints are used and multiple subjects meet the constraints, the subject will be selected randomly from the set of matching subjects.

6.1.1 Configuration

The configuration for the ANSUR Handler specifies which subject will be used in the analysis. Selection is controlled by applying one or more constraints. There are three constraint types: (1) **`StringConstraint`**, (2) **`IntegerConstraint`**, and (3) **`DoubleConstraint`**. These constraints filter string values, integer values, and floating-point values, respectively. However, because the ANSUR data set does not include floating-point data, the **`DoubleConstraint`** type is not used in practice.

The JSON form of a **`StringConstraint`** is shown in Code Excerpt 16. The ***`propertyName`*** key at Line 1 specifies to which property the constraint is to be applied. The property names correspond to the column headers in the ANSUR data sets and can also be found in the **`ValueKey`** enumeration. The ***`matches`*** key on Line 2 specifies a regular expression pattern that is to be used to check for a match.

Code Excerpt 16. Example JSON for an ANSUR StringConstraint

```
{
  "class" : "mil.devcom_sc.ansur.api.constraints.StringConstraint",
  "propertyName": "Gender",                                          // 1
  "matches": "[Mm][Aa][Ll][Ee]"                                      // 2
}
```

Examples of **`IntegerConstraints`** are shown in Code Excerpts 17 and 18. Within an **`IntegerConstraint`** there are three possible arguments: ***`lowerBound`***, ***`upperBound`***, and ***`equals`***. The ***`lowerBound`*** argument specifies the minimum value of the property specified by the ***`propertyName`*** argument. Similarly, ***`upperBound`*** specifies the maximum value of the property. Both can be used together to bracket a range as was done in Code Excerpt 17. If an ***`equals`*** argument is used, as in Code Excerpt 18, the value of the property must match exactly.

Code Excerpt 17. Example JSON for an ANSUR IntegerConstraint with Upper and Lower Bounds

```
{
  "class" : "mil.devcom_sc.ansur.api.constraints.IntegerConstraint",
  "propertyName": "weightkg",
  "lowerBound" : 550
  "upperBound" : 650
}
```

Code Excerpt 18. Example JSON for an ANSUR IntegerConstraint with an Equals Argument

```
{
  "class" : "mil.devcom_sc.ansur.api.constraints.IntegerConstraint",
  "propertyName": "subjectID",
  "equals" : 10173
}
```

Multiple constraints can be applied. The full form of the ANSUR configuration is shown in Code Excerpt 19. One bit of formatting to note is the ***`constraints`*** parameter at Line 1. It is required to wrap the **`StringConstraint`** and **`IntegerConstraint`** blocks in the constraints list even if there is only one constraint.

Code Excerpt 19. ANSUR Configuration with Multiple Constraints

```
{
  "class" : "mil.devcom_sc.ansur.api.ANSURConfiguration",
  "constraints": [                                                    // 1
    {
      "class" : "mil.devcom_sc.ansur.api.constraints.StringConstraint",
      "propertyName": "Gender",
      "matches": "Male"
    },
    {
      "class" : "mil.devcom_sc.ansur.api.constraints.IntegerConstraint",
      "propertyName": "balloffootlength",
      "equals": 198
    },
    {
      "class" : "mil.devcom_sc.ansur.api.constraints.IntegerConstraint",
      "propertyName": "footlength",
      "equals": 270
    },
    {
      "class" : "mil.devcom_sc.ansur.api.constraints.IntegerConstraint",
      "propertyName": "lowerthighcircumference",
      "lowerBound": 390,
      "upperBound": 395
    },
    {
      "class" : "mil.devcom_sc.ansur.api.constraints.StringConstraint",
      "propertyName": "SubjectsBirthLocation",
      "matches": ".*tuck.*"
    }
  ]
}
```

6.1.2 Commands and Responses

The ANSUR handler supports one command message: **`GetValueMessage`**. The JSON form for **`GetValueMessage`** is shown in Code Excerpt 20. The value of the ***`valueKey`*** parameter at Line 1 can be the name of any of the ANSUR metrics as defined in the **`ValueKey`** enumeration. The possible values are listed in the source code in Appendix A.

Note the inconsistency between the **propertyName** key used in the configuration and the **valueKey** used in the **GetValueMessage**. This inconsistency should be addressed in a future version of the ANSUR plugin.

Code Excerpt 20. Sample JSON for ANSUR GetValueMessage

```
{
  "class" : "mil.devcom_sc.ansur.messages.GetValueMessage",
  "valueKey" : “GENDER”                                          // 1
}
```

The response for a **GetValueMessage** is a **GetValueResponse**. The JSON form for a **GetValueResponse** is shown in Code Excerpt 21. At Line 1, the **ValueKey** from the query is echoed back in the response. The values at Lines 2, 3, and 4 will be populated based on the type of the value. If the property that was queried is an integer, then the ***intValue*** property will be set to the actual value, the ***doubleValue*** property will be set to **Double.MIN_VALUE,** and the ***stringValue*** property will be null. If the property that was queried is a string, the ***stringValue*** property will be set, the ***intValue*** property will be **Integer.MIN_VALUE,** and the ***doubleValue*** property will be **Double.MIN_VALUE**. Admittedly, this approach is clumsy and convoluted and will likely change in the future.

Code Excerpt 21. Example JSON for ANSUR GetValueResponse

```
{
  "class" : "mil.devcom_sc.ansur.messages.GetValueMessage",
  "valueKey" : “ValueKey”                                        // 1
  "intValue" : integer value                                     // 2
  "doubleValue" : double value                                   // 3
  "stringValue" : string value                                   // 4
}
```

6.2 Equipment Handler

The Equipment **`Handler`** tracks what the Soldier is carrying and how much that equipment weighs. The handler currently supports three types of items: (1) packs, (2) guns, and (3) magazines.

Each item has a calculated mass that includes anything attached to or in it. The mass is dynamic and can change over time. For example, the mass of the gun includes the mass of its magazine and the mass of the rounds in the magazine. Each time the gun is shot, its mass is decremented.

The Soldier can have an arbitrary amount of equipment. The set of all the equipment carried by a Soldier is commonly referred to as the Soldier's "kit." The object model for representing Soldier equipment is shown in Figure 7.

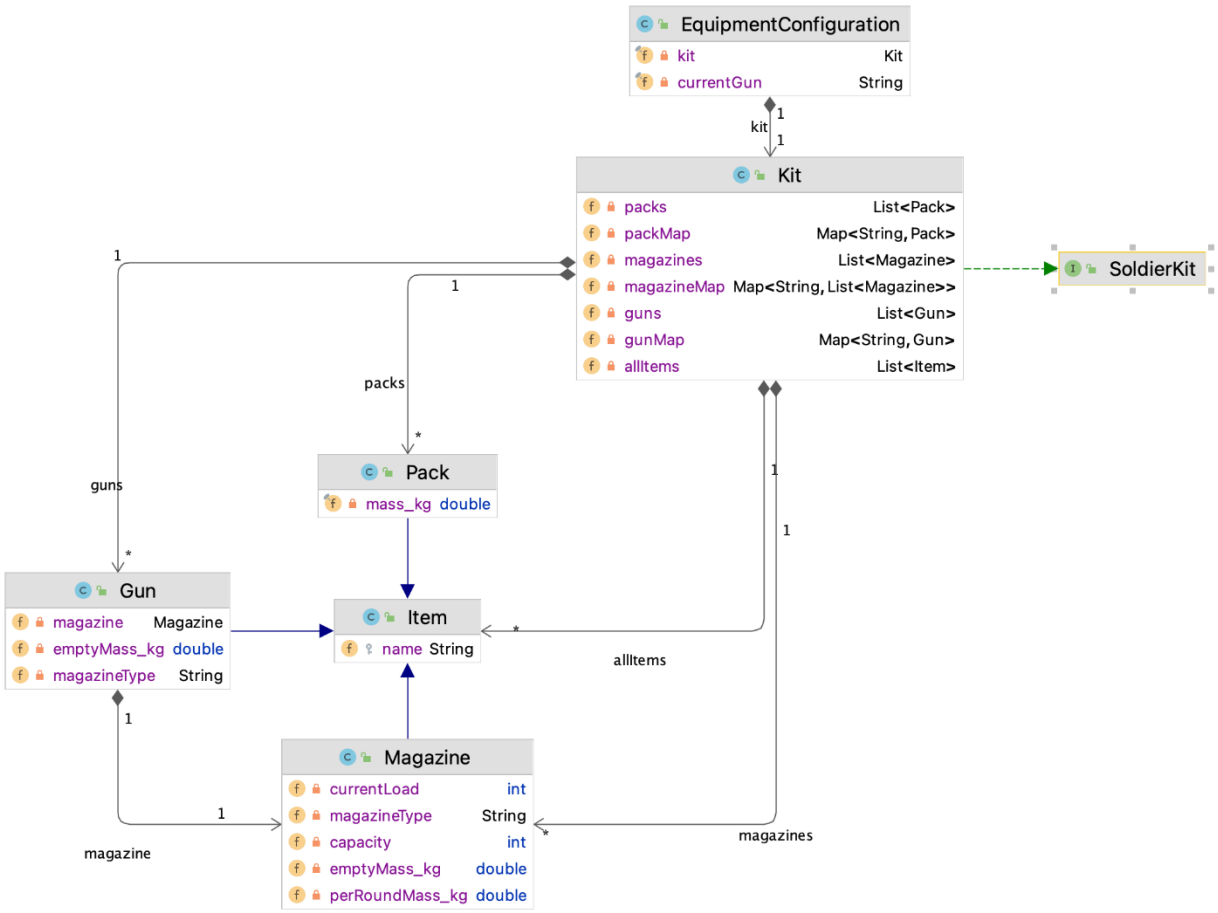

Figure 7. Object model for Soldier equipment

6.2.1 Configuration

The configuration for the Equipment **`Handler`** specifies the equipment that the Soldier is carrying. The JSON for the equipment configuration mirrors the object model. An example of the JSON for the **`EquipmentConfiguration`** is shown in Code Excerpt 22. This is the object that is included in the ***`configurations`*** block in the Soldier definition. The EquipmentConfiguration has two fields. At Line 1 is the definition for the Soldier's kit. As in all other JSON configurations, file references are allowed here. Line 2 specifies which gun the Soldier will use. If ***`currentGun`*** is unspecified, the Soldier will use the first gun in his collection.

Code Excerpt 22. Example JSON for EquipmentConfiguration

```
{
  "class" : "mil.devcom_dac.equipment.api.EquipmentConfiguration",
  "kit" : "TestKit.json"                                          // 1
  "currentGun" : "M16A1"                                          // 2
}
```

An example of the Soldier's kit definition is shown in Code Excerpt 23. The list of guns that the Soldier is carrying is indicated by the gun's parameter at Line 1. At Line 2 is the list of magazines, and at Line 3 is the list of packs.

Code Excerpt 23. Example JSON for Soldier Kit Configuration

```
{
  "class" : "mil.devcom_dac.equipment.api.Kit",
  "guns": [                                                    // 1
    "M16A1.json"
  ],
  "magazines": [                                               // 2
    "STANAGMagazine.json",
    "STANAGMagazine.json",
    "STANAGMagazine.json",
    "STANAGMagazine.json"
  ],
  "packs": [                                                   // 3
    "FannyPack.json"
  ]
}
```

The configuration for a gun is shown in Code Excerpt 24. The name of the gun is specified with the ***name*** parameter. This name is used to specify which gun is used when shooting. Gun names must be unique within a Soldier's kit. At Line 2 is the specification for the type of magazine that the gun accepts. The ***magazineType*** parameter expresses compatibility. Any magazine of the specified type can be loaded into the gun. The ***emptyMass_kg*** parameter at Line 3 specifies the mass of the gun without a magazine or rounds in it. The total mass of the gun is calculated dynamically during the simulation based on the magazine and number of rounds.

Code Excerpt 24. Example JSON for Gun Definition

```
{
  "class" : "mil.devcom_dac.equipment.api.Gun",
  "name": "M16A1",                                             // 1
  "magazineType": "5.56mm STANAG",                             // 2
  "emptyMass_kg": 5.5                                          // 3
}
```

The configuration for a magazine is shown in Code Excerpt 25. At Line 1 is the ***name*** of the magazine. This name is used as a label and does not need to be unique. The ***magazineType*** parameter at Line 2 specifies the compatibility of the magazine. Multiple magazines can be of the same type. For example, one could have magazines with different capacities. Each would have the same ***magazineType*** but different names. The

capacity parameter specifies the maximum number of rounds that the magazine can hold and the ***currentLoad*** parameter specifies how many rounds are in it at the moment. The ***perRoundMass_kg*** parameter at Line 5 specifies how much each round weighs and the ***emptyMass_kg*** parameter specifies how much the magazine weighs without rounds in it.

Code Excerpt 25. Example JSON for Magazine Definition

```
{
  "class" : "mil.devcom_dac.equipment.api.Magazine",
  "name": "5.56mm STANAG 30rd",                                    // 1
  "magazineType": "5.56mm STANAG",                                 // 2
  "capacity": 30,                                                  // 3
  "currentLoad" : 30,                                              // 4
  "perRoundMass_kg": 0.01231,                                      // 5
  "emptyMass_kg": 0.1207                                           // 6
}
```

The final item type that can be added to the Soldier is a pack. An example of the JSON for defining a pack is shown in Code Excerpt 26. The definition for a pack is simple. The name of the pack is specified with the name parameter as on Line 1, and the total mass of the pack is specified by the ***mass_kg*** parameter as on Line 2. Note that the name of a pack must also be unique for a Soldier. There is currently no ability to add items to a pack.

Code Excerpt 26. Example JSON for Pack Definition

```
{
  "class" : "mil.devcom_dac.equipment.api.Pack",
  "name": "Fanny Pack",                                            // 1
  "mass_kg": 1.5                                                   // 2
}
```

It is expected that the **`EquipmentHandler`** will need to be enhanced to meet the requirements of analysts. The model currently has no explicit accounting for radios, clothing, or armor. In the near term, such things can be declared as packs to get the total Soldier load correct. However, this is an inadequate solution. Ideally, the **`EquipmentHandler`** will be able to provide all the information necessary to enable the human dynamics model to predict motion. This will require the handler to provide moments of inertia and carry location.

6.2.2 Commands and Responses

The Equipment Handler supports the four commands shown in the object model in Figure 8. The responses for the commands are shown in the object model in Figure 9. The **`GetInventory`** command produces the Inventory response. The **`SetGun`**, **`Shoot`**, and **`Reload`** commands all produce a **`GunState`**.

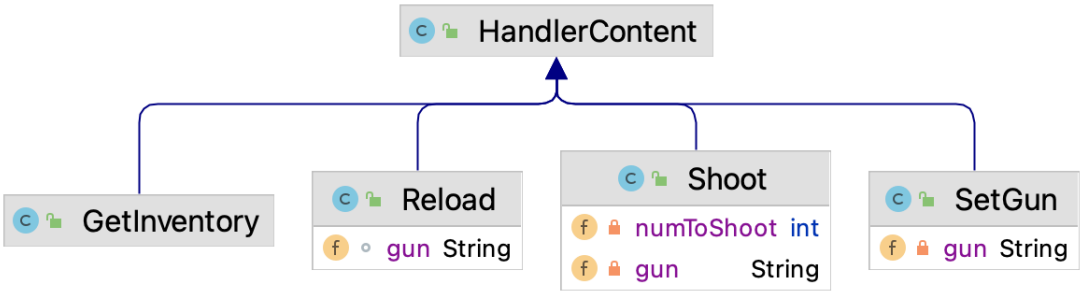

Figure 8. Object model for equipment handler commands

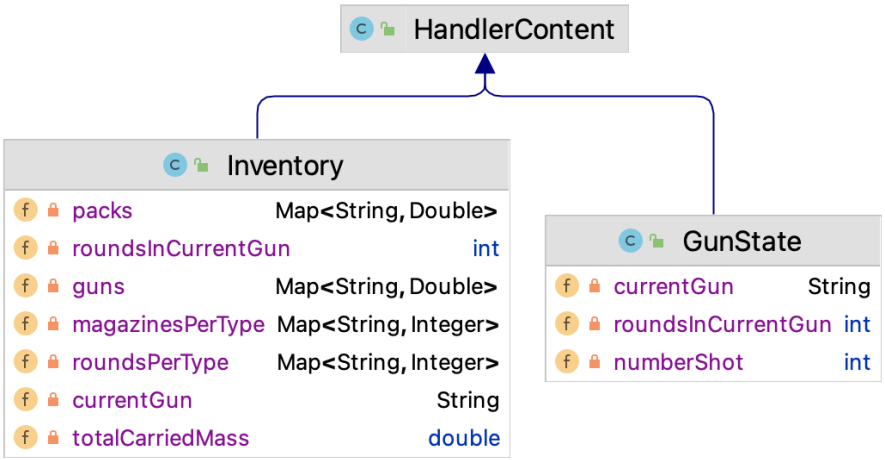

Figure 9. Object model for equipment handler responses

The **`GetInventory`** command requests a full inventory of the Soldier's kit. Since the **`GetInventory`** class has no parameters, its JSON form is simple. It is shown in Code Excerpt 27. On the other hand, the resulting **`Inventory`** class contains several parameters. An example of its JSON is shown in Code Excerpt 28. The ***`currentGun`*** parameter at Line 1 contains the name of the gun that the Soldier is currently using. The ***`roundsInCurrentGun`*** parameter at Line 2 contains the number of rounds remaining in the magazine in the current gun. At Line 3, the ***`totalCarriedMass`*** parameter expresses the total mass of the Soldier's kit. The ***`packs`*** object that starts at Line 4 contains a map of the Soldier's packs and their mass. Similarly, the ***`guns`*** object at Line 5 contains a

map of the Soldier's guns and their current mass. Note that this includes the mass of the loaded magazine. At Line 6, ***magazinesByType*** is a map of the number of magazines of each type. The magazine count includes the loaded magazines. The count of remaining rounds per magazine type is found at Line 7. The remaining round count also includes the rounds remaining in each gun.

Code Excerpt 27. Example JSON for GetInventory Object

```
{
  "class" : "mil.devcom_dac.equipment.messages.GetInventory"
}
```

Code Excerpt 28. Example JSON for Inventory Object

```
{
  "class" : "mil.devcom_dac.equipment.messages.Inventory",
  "currentGun" : "M5",                                          // 1
  "roundsInCurrentGun" : 12,                                    // 2
  "totalCarriedMass" : 45.34,                                   // 3
  "packs" : {                                                   // 4
    "Backpack" : 34.21,
  }
  "guns" : {                                                    // 5
    "M5" : 7.25,
    "M17" : 1.89,
  }
  "magazinesPerType" : {                                        // 6
    "M5" : 8,
    "M17" : 2,
  }
  "roundsPerType" : {                                           // 7
    "M5" : 152,
    "M17" : 42,
  }
}
```

The **SetGun**, **Shoot**, and **Reload** messages control interaction with the Soldier's weapon. The **SetGun** message sets the gun with the specified name as the current weapon for the Soldier. The JSON for this command is shown in Code Excerpt 29. The **Shoot**

message commands the Soldier to fire one or more rounds. Its JSON is shown in Code Excerpt 30. If the ***gun*** parameter is not included, the current gun is used. If the gun specified is not the current gun, the specified gun becomes the current gun. The number of rounds to fire is specified by the ***numToShoot*** parameter. If a value is not provided, one round will be fired. If the number of rounds available is less than the number available, the available rounds will be fired and that number returned in the **GunState** object.

The **Reload** command instructs the Soldier to replace the magazine in the specified gun. The **Reload** command behaves similarly to the **Shoot** command with respect to the value of the ***gun*** parameter. Its JSON is shown in Code Excerpt 31. Reloads are not automatic. They must be explicitly commanded.

Code Excerpt 29. Example JSON for SetGun Command

```
{
  "class" : "mil.devcom_dac.equipment.messages.SetGun",
  "gun" : "M5"
}
```

Code Excerpt 30. Example JSON for Shoot Command

```
{
  "class" : "mil.devcom_dac.equipment.messages.Shoot",
  "gun" : "M5",
  "numToShoot" : 3
}
```

Code Excerpt 31. Example JSON for Reload Command

```
{
  "class" : "mil.devcom_dac.equipment.messages.Reload",
  "gun" : "M5"
}
```

The result of the **SetGun**, **Shoot,** and **Reload** commands is a **GunState** object. The JSON for the **GunState** is shown in Code Excerpt 32. The ***numberShot*** parameter contains the number of rounds that were fired as the result of a **Shoot** command. It will be zero for the **SetGun** and **Reload** commands.

Code Excerpt 32. Example JSON for GunState Object

```
{
  "class" : "mil.devcom_dac.equipment.messages.GunState",
  "currentGun" : "M5",
  "numberShot" : 0,
  "roundsInCurrentGun" : 20
}
```

6.3 Telemetry Agent

The Telemetry **`Agent`** records entity data to a comma-separated value (CSV) file. The data that is written is taken from annotated objects that have been cached in the entity's **`Blackboard`**. The data objects may have multiple fields, each of which is marked with a **`@StateProperty`** annotation. Each marked field is recorded as a column in the CSV file. Data objects are selected from the **`Blackboard`** by using the key associated with the object. Recordable objects and their keys should be noted in the documentation for each feature.

All telemetry data is recorded under a single top-level directory. The directory is specified using the ***`mil.sstaf.telemetry.outputDir`*** JVM property. Under that top-level directory, the entity's path is replicated as a directory hierarchy. Within the directory for the entity itself, separate CSV files are created for each recorded object. The name of the key serves as the name of the file.

The data is written for each object on every tick of the simulation.

6.3.1 Configuration

The configuration for the Telemetry agent is simply the list of keys for the objects to be recorded. The example JSON for the configuration is shown in Code Excerpt 33.

Code Excerpt 33. Example JSON for Telemetry Configuration

```
{
  "class" : "mil.devcom_dac.equipment.messages.GunState",
  "stateKeys" : [
      "muscleStates",
      "aimError",
      …
  ]
}
```

6.3.2 Commands and Responses

The Telemetry agent does not support any interactive commands.

7. SSTAF SOURCE CODE

The source code for the SSTAF system is included in the appendices. The code is composed of three components. The first component is the code for the SSTAF system, which is included in Appendix A. The SSTAF system code includes the core framework, several `Feature` implementations, relevant test cases, and test data. The second component is the code for the SSTAF Gradle plugin, which is an extension for the Gradle build system that handles special requirements for building SSTAF. It is in Appendix B. The third component is in Appendix C and is a small application that can be used to reassemble the source code files for the system and SSTAF Gradle plugin from flat text files.

To reassemble the source code from this report:

1. Extract the contents of Appendices A and B to separate text files, making certain to keep the file labels intact.
2. Compile the simple Java application from Appendix C.
3. Run the Appendix C application on the system code from Appendix A with the first argument being the file name for the system code, the second argument being the destination directory for the generated source tree, and the third argument being "***`SSTAF/src`***".
4. Run the application on the SSTAF Gradle plugin code from Appendix B with the first argument being the file name for the plugin code, the second argument being the destination directory for the generated source tree, and the third argument being "***`sstafGradlePlugin/`***".
5. Obtain the ANSUR II data from DEVCOM Soldier Center or download it from https://www.openlab.psu.edu/ansur2/. Install the male and female data sets in ***`SSTAF/src/features/ansurHandler/mil.devcom_sc.ansur.handler/src/main/resources/ansur`***. The file for the male subject data should be named "ANSUR II MALE Public.csv", and "ANSUR II FEMALE Public.csv" for the female data.
6. Compile and install the SSTAF Gradle plugin using "***`gradle build publishToMavenLocal`***".
7. Compile and install the SSTAF system using "***`gradle build`***".

8. CONCLUSION

The SSTAF Interactive Analyzer provides flexibility for both software developers and analysts. It enables integration with applications that are implemented in non-JVM languages, thus freeing developers to create client applications in the language of their choice. It also enables executing an SSTAF analysis through a script. While the JSON-based scripts are awkward, the ability to script SSTAF analyses provides a significant increase in the usability of SSTAF as an analysis environment.

Now that the core architecture and functionality of SSTAF is complete, future work will focus on increasing the number, quality, and robustness of the human models. By publishing as much of the source code as is possible, it is hoped that others might take an interest in SSTAF and contribute models or enhancements to the system.

9. REFERENCES

Bowers, R. A. (2021). *The Soldier and squad trade space analysis framework* (DEVCOM DAC-TR-2021-007). DEVCOM Analysis Center.

Bloch, J. (2008). *Effective Java* (2nd ed.). Pearson Education.

Appendix A – Source Code for SSTAF

A.1 SSTAF/src/README.md

```
# Introduction
This directory includes the source code, resources, test cases and build scripts for the core com
ponents of the Soldier
and Squad Trade-space Analysis Framework (SSTAF).
# Project Structure
This project is organized as shown in Table 1. The only artifacts intended for use in production
are found in the "
framework" and "features" directories.
Table 1: Subdirectories
| Directory | Description  |
|:----------|:----------|
| build     | Build artifacts (gradle convention)  |
| buildSrc  | Source code for custom Gradle components   |
| features  | Source, test and resources for plugin feature modules that are <br> included with t
he core system |
| framework | Source, test and resources for the core SSTAF modules |
| testFeatures | Source for plugin modules used for testing. These features <br> should not be us
ed for production runs.  |
# Building SSTAF
## Java
SSTAF requires Java Development Kit (JDK) version 11 or later. SSTAF uses Java 11 language featur
es and cannot be run on
an earlier version without significant changes.
SSTAF does not use any dangerous or deprecated constructs such as the sun.misc.Unsafe class, so i
t should compile and
run without issue on newer JDKs. Future work will modernize SSTAF in accordance with new language
constructs.
## Gradle
The minimum Gradle version that has been used to build and test SSTAF is Gradle 6.7.1. Gradle 6.4
might be sufficient
since it is advertised to have full support for Java modules (Jigsaw), but this has not been test
ed.
```

A.2 SSTAF/src/applications/analyzer/build.gradle

```
plugins {
    id 'sstaf'
    id 'jvm-test-suite'
    id 'maven-publish'
    id 'application'
}
repositories {
    mavenCentral()
    mavenLocal()
}
dependencies {
    implementation project(':framework:mil.sstaf.core')
    implementation project(':framework:mil.sstaf.session')
    implementation project(':framework:mil.sstaf.analyzer')
    implementation group: 'org.apache.commons', name: 'commons-math3', version: '3.6.1'
    implementation group: 'org.slf4j', name: 'slf4j-api', version: '1.7.32'
    implementation 'org.junit.jupiter:junit-jupiter:5.8.1'
    runtimeOnly group: 'org.apache.commons', name: 'commons-csv', version: '1.8'
    runtimeOnly group: 'org.slf4j', name: 'slf4j-simple', version: '1.7.32'
    runtimeOnly group: 'net.java.dev.jna', name: 'jna', version: '5.4.0'
    runtimeOnly project(':features:ansurHandler:mil.devcom_sc.ansur.api')
    runtimeOnly project(':features:ansurHandler:mil.devcom_sc.ansur.messages')
    runtimeOnly project(':features:ansurHandler:mil.devcom_sc.ansur.handler')
    runtimeOnly project(':features:support:mil.sstaf.telemetry')
    runtimeOnly project(':features:equipment:mil.devcom_dac.equipment.api')
    runtimeOnly project(':features:equipment:mil.devcom_dac.equipment.handler')
    runtimeOnly project(':features:simplePhysiologyAgent:mil.sstaf.physiology.agent')
    runtimeOnly project(':features:simplePhysiologyAgent:mil.sstaf.physiology.api')
    runtimeOnly project(':features:simplePhysiologyAgent:mil.sstaf.physiology.models.api')
    runtimeOnly project(':features:simplePhysiologyAgent:mil.sstaf.physiology.models.cardiovascul
ar')
    runtimeOnly project(':features:simplePhysiologyAgent:mil.sstaf.physiology.models.cognition')
    runtimeOnly project(':features:simplePhysiologyAgent:mil.sstaf.physiology.models.energy')
    runtimeOnly project(':features:simplePhysiologyAgent:mil.sstaf.physiology.models.hydration')
    runtimeOnly project(':features:simplePhysiologyAgent:mil.sstaf.physiology.models.musculature'
)
    runtimeOnly project(':features:simplePhysiologyAgent:mil.sstaf.physiology.models.respiration'
)
    runtimeOnly project(':features:simplePhysiologyAgent:mil.sstaf.physiology.models.vision')
    runtimeOnly project(':features:support:mil.sstaf.blackboard.api')
    runtimeOnly project(':features:support:mil.sstaf.blackboard.inmem')
}
sstaf {
    // This does not have to be a complete description (e.g. here 'org.apache.commons.collections
' does not export anything here).
    // It only needs to be good enough to work in the context of this application we are building
.
    module('commons-math3-3.6.1.jar', 'commons.math3', '3.6.1') {
        exports('org.apache.commons.math3.random')
        exports('org.apache.commons.math3.analysis.function')
        exports('org.apache.commons.math3.analysis.polynomials')
        exports('org.apache.commons.math3.analysis')
    }
    module('jopt-simple-4.6.jar', 'jopt.simple', '4.6')
    module('jmh-generator-annprocess-1.21.jar', 'jmh.generator.annprocess', '1.21')
    module('jmh-core-1.21.jar', 'jmh.core', '1.21') {
        exports('org.openjdk.jmh.annotations')
        exports('org.openjdk.jmh')
    }
    //
    // CSV support for ANSUR
    //
    module('commons-csv-1.8.jar', 'commons.csv', '1.8') {
        exports 'org.apache.commons.csv'
```

```
    }
    //
    // Set the hashing algorithm to be used for the resources.
    //
    hashAlgorithm = 'SHA-384'
}

application {
    mainClass = 'mil.sstaf.analyzer.Main'
    mainModule = 'mil.sstaf.analyzer'
    applicationName = 'sstaf-analyzer'
    applicationDefaultJvmArgs = ['--add-modules', 'ALL-MODULE-PATH']
}
testing {
    suites {
        integrationTest(JvmTestSuite) {
            dependencies {
                implementation project(':framework:mil.sstaf.core')
                implementation project(':framework:mil.sstaf.session')
                implementation project(':framework:mil.sstaf.analyzer')
                runtimeOnly project(':features:ansurHandler:mil.devcom_sc.ansur.api')
                runtimeOnly project(':features:ansurHandler:mil.devcom_sc.ansur.handler')
                runtimeOnly project(':features:ansurHandler:mil.devcom_sc.ansur.messages')
                runtimeOnly project(':features:equipment::mil.devcom_dac.equipment.handler')
                runtimeOnly project(':features:equipment:mil.devcom_dac.equipment.api')
                runtimeOnly project(':features:equipment:mil.devcom_dac.equipment.handler')
                runtimeOnly project(':features:simplePhysiologyAgent::mil.sstaf.physiology.api')
                runtimeOnly project(':features:simplePhysiologyAgent:mil.sstaf.physiology.agent')
                runtimeOnly project(':features:simplePhysiologyAgent:mil.sstaf.physiology.api')
                runtimeOnly project(':features:simplePhysiologyAgent:mil.sstaf.physiology.models.
api')
                runtimeOnly project(':features:simplePhysiologyAgent:mil.sstaf.physiology.models.
cardiovascular')
                runtimeOnly project(':features:simplePhysiologyAgent:mil.sstaf.physiology.models.
cognition')
                runtimeOnly project(':features:simplePhysiologyAgent:mil.sstaf.physiology.models.
energy')
                runtimeOnly project(':features:simplePhysiologyAgent:mil.sstaf.physiology.models.
hydration')
                runtimeOnly project(':features:simplePhysiologyAgent:mil.sstaf.physiology.models.
musculature')
                runtimeOnly project(':features:simplePhysiologyAgent:mil.sstaf.physiology.models.
respiration')
                runtimeOnly project(':features:simplePhysiologyAgent:mil.sstaf.physiology.models.
vision')
                runtimeOnly project(':features:support:mil.sstaf.blackboard.api')
                runtimeOnly project(':features:support:mil.sstaf.blackboard.api');
                runtimeOnly project(':features:support:mil.sstaf.blackboard.inmem')
                runtimeOnly project(':features:support:mil.sstaf.telemetry')
            }
            targets {
                all {
                    testTask.configure {
                        shouldRunAfter(test)
                    }
                }
            }
        }
    }
}
```

A.3 SSTAF/src/applications/analyzer/src/integrationTest/java/analyzer/SSTAFAnalyzerIntegrationTest.java

```
package analyzer;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Nested;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.MethodSource;
import java.io.*;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;
import static org.junit.jupiter.api.Assertions.*;
public class SSTAFAnalyzerIntegrationTest {
    static final String endSessionMsg = "{ \"class\" : \"mil.sstaf.analyzer.commands.EndSession\"
}\n";
    static final String getEntitiesMsg = "{ \"class\" : \"mil.sstaf.analyzer.commands.GetEntities
\"}\n";
    static final boolean HEAVY = true; // Boolean.getBoolean("HEAVY_TEST");
    static Path extractedPath;
    static Path extractedLibPath;
    static Path userDir;
    static Path baseDir;
    static Path inputDir;
    static Path resourceDir;
    /**
     * Unzip the zip file to the destination directory
     *
     * @param zipFilePath
     * @param destDir
     */
    static void unzip(Path zipFilePath, Path destDir) throws IOException {
        File dir = destDir.toFile();
        // create output directory if it doesn't exist
        if (!dir.exists()) {
            Files.createDirectories(destDir);
        }
        FileInputStream fis;
        //buffer for read and write data to file
        byte[] buffer = new byte[1024];
        try {
            fis = new FileInputStream(zipFilePath.toFile());
            ZipInputStream zis = new ZipInputStream(fis);
            ZipEntry ze = zis.getNextEntry();
            while (ze != null) {
                String fileName = ze.getName();
                Path entryPath = Path.of(destDir.toString(), fileName);
                System.out.println("Unzipping to " + entryPath.normalize().toAbsolutePath());
                if (ze.isDirectory()) {
                    Files.createDirectories(entryPath);
                } else {
                    File newFile = entryPath.toFile();
                    FileOutputStream fos = new FileOutputStream(newFile);
                    int len;
                    while ((len = zis.read(buffer)) > 0) {
                        fos.write(buffer, 0, len);
                    }
                    fos.close();
                }
```

```
                //close this ZipEntry
                zis.closeEntry();
                ze = zis.getNextEntry();
            }
            //close last ZipEntry
            zis.closeEntry();
            zis.close();
            fis.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    static Path extractDistribution() throws IOException {
        String cwdString = System.getProperty("user.dir");
        Path analyzerZip = Path.of(cwdString,
                "build", "distributions", "sstaf-analyzer.zip");
        Path tmpDir = Path.of(cwdString, "build", "tmp");
        Path expandedDir = Files.createTempDirectory(tmpDir, "SSTAFAnalyzerTest");
        unzip(analyzerZip.normalize().toAbsolutePath(),
                expandedDir.normalize().toAbsolutePath());
        return Path.of(expandedDir.toString(), "sstaf-analyzer");
    }
    @BeforeAll
    static void beforeAll() {
        try {
            userDir = Path.of(System.getProperty("user.dir"));
            baseDir = Path.of(userDir.toString(), "..", "..");
            inputDir = Path.of(baseDir.toString(), "testInput");
            resourceDir = Path.of(userDir.toString(), "src",
                    "integrationTest", "resources");
            extractedPath = extractDistribution();
            extractedLibPath = Path.of(extractedPath.toString(), "lib").normalize().toAbsolutePat
h();
        } catch (IOException e) {
            fail("Could not extract distribution");
        }
    }

    private static List<String> getGoodEntityFiles() {
        List<String> files;
        if (HEAVY) {
            files = List.of(
                    Path.of(inputDir.toString(), "goodEntityFiles", "OneSoldier.json").toString()
,
                    Path.of(inputDir.toString(), "goodEntityFiles", "FourSoldiers.json").toString
(),
                    Path.of(inputDir.toString(), "goodEntityFiles", "OneFireTeam.json").toString(
),
                    Path.of(inputDir.toString(), "goodEntityFiles", "OneSquad.json").toString(),
                    Path.of(inputDir.toString(), "goodEntityFiles", "TwoSquads.json").toString(),
                    Path.of(inputDir.toString(), "goodEntityFiles", "OnePlatoon.json").toString()
,
                    Path.of(inputDir.toString(), "goodEntityFiles", "BlueRedGray.json").toString(
)
            );
        } else {
            files = List.of(
                    Path.of(inputDir.toString(), "goodEntityFiles", "OneSoldier.json").toString()
            );
        }
        return files;
    }
    String makeModulePath() {
        StringBuilder sb = new StringBuilder();
        sb.append("--module-path=");
        sb.append(extractedLibPath.toString());

        List<String> modulePaths = List.of();
```

```
        for (Iterator<String> iter = modulePaths.iterator(); iter.hasNext(); ) {
            String module = iter.next();
            Path path = Path.of(userDir.toString(), "..", "..", module, "build", "libs").normaliz
e().toAbsolutePath();
            sb.append(path);
            if (iter.hasNext()) sb.append(":");
        }
        return sb.toString();
    }
    private Process makeProcessWithArg(String arg) throws IOException {
        ProcessBuilder processBuilder = makeDefaultProcessBuilder(arg);
        return processBuilder.start();
    }
    private Process makeProcessWithNoArgs() throws IOException {
        ProcessBuilder processBuilder = makeDefaultProcessBuilder(null);
        return processBuilder.start();
    }
    private ProcessBuilder makeDefaultProcessBuilder(String arg) {
        ProcessBuilder processBuilder = new ProcessBuilder();
        String modulePath = makeModulePath();
        List<String> commandElements = new ArrayList<>();
        commandElements.add("java");
        commandElements.add(modulePath);
        commandElements.add("--module");
        commandElements.add("mil.sstaf.analyzer/mil.sstaf.analyzer.Main");
        if (arg != null) {
            commandElements.add(arg);
        }
        processBuilder.command(commandElements);
        processBuilder.redirectError(ProcessBuilder.Redirect.INHERIT);
        return processBuilder;
    }
    @ParameterizedTest(name = "{index} ==> Testing with entity file ''{0}''")
    @MethodSource("getGoodEntityFiles")
    @DisplayName("Confirm that good entity files can be loaded")
    void goodFiles(String entityFile) {
        assertDoesNotThrow(() -> {
            Process p = makeProcessWithArg(entityFile);
            Thread.sleep(2000);
            OutputStreamWriter osw = new OutputStreamWriter(p.getOutputStream());
            BufferedWriter writer = new BufferedWriter(osw);
            InputStreamReader isr = new InputStreamReader(p.getInputStream());
            BufferedReader reader = new BufferedReader(isr);
            System.out.println("Sending " + getEntitiesMsg);
            writer.write(getEntitiesMsg);
            writer.flush();
            while (true) {
                String in = reader.readLine();
                if (in != null) {
                    System.out.printf("Received - %s\n", in);
                    break;
                } else {
                    Thread.sleep(100);
                }
            }
            System.out.println("Writing " + endSessionMsg);
            writer.write(endSessionMsg);
            writer.flush();
            boolean gotExitMessage = false;
            while (p.isAlive()) {
                String in2 = reader.readLine();
                System.out.println("Read " + in2);
                if (in2 != null) {
                    System.out.printf("Read - %s\n", in2);
                    if (in2.contains("EndSession")) {
                        gotExitMessage = true;
                    }
                } else {
```

```
                    Thread.sleep(100);
                }
            }
            assertEquals(0, p.exitValue());
            assertTrue(gotExitMessage);
        });
    }

    private static List<String> getBadEntityFiles() {
        List<String> files = List.of(
                Path.of(inputDir.toString(), "badEntityFiles", "emptyFile.json").toString(),
                Path.of(inputDir.toString(), "badEntityFiles", "somethingOtherThanEntity.json").t
oString()
        );
        return files;
    }
    @Nested
    @DisplayName("Test start-up failure modes")
    class FailureTests {
        @Test
        @DisplayName("Confirm that an Analyzer started without arguments runs but exits with code
255.")
        void testOne() {
            assertDoesNotThrow(() -> {
                Process p = makeProcessWithNoArgs();
                while (p.isAlive()) {
                    Thread.sleep(1000);
                }
                assertEquals(1, p.exitValue());
            });
        }
        @ParameterizedTest(name = "{index} ==> Testing with entity file ''{0}''")
        @MethodSource("analyzer.SSTAFAnalyzerIntegrationTest#getBadEntityFiles")
        @DisplayName("Confirm that bad entity files result in an immediate exit with code 1")
        void badFiles(String entityFile) {
            assertDoesNotThrow(() -> {
                Process p = makeProcessWithArg(entityFile);
                while (p.isAlive()) {
                    Thread.sleep(1000);
                }
                assertEquals(1, p.exitValue());
            });
        }
    }
    private String makeEntityCommand(String command) {
        return "{ \"class\" : \"mil.sstaf.analyzer.commands.Command\", " +
                "\"path\" : \"Test Platoon:Squad C:Fire Team Bravo:AR\", " +
                "\"content\" : {" +
                command +
                "} }\n";
    }
    @DisplayName("Confirm that messages can be sent and results received")
    @Nested
    class MessageTests {
        @Test
        @DisplayName("Confirm ANSUR query works")
        void messageHandlingTest() {
            assertDoesNotThrow(() -> {
                String entityFile = Path.of(inputDir.toString(), "goodEntityFiles", "OnePlatoon.j
son").toString();
                Process p = makeProcessWithArg(entityFile);
                Thread.sleep(2000);
                OutputStreamWriter osw = new OutputStreamWriter(p.getOutputStream());
                BufferedWriter writer = new BufferedWriter(osw);
                InputStreamReader isr = new InputStreamReader(p.getInputStream());
                BufferedReader reader = new BufferedReader(isr);
                System.out.println("Sending " + getEntitiesMsg);
                writer.write(getEntitiesMsg);
```

```
                writer.flush();
                while (true) {
                    String in = reader.readLine();
                    if (in != null) {
                        System.out.printf("Received - %s\n", in);
                        break;
                    } else {
                        Thread.sleep(100);
                    }
                }
                String query = makeEntityCommand(
                        "\"class\" : \"mil.devcom_sc.ansur.messages.GetValueMessage\", \"key\" :
\"EAR_LENGTH\""
                );
                writer.write(query);
                writer.flush();
                System.out.println("Writing " + query);
                while (true) {
                    String in = reader.readLine();
                    if (in != null) {
                        System.out.printf("Received - %s\n", in);
                        break;
                    } else {
                        Thread.sleep(100);
                    }
                }
                System.out.println("Writing " + endSessionMsg);
                writer.write(endSessionMsg);
                writer.flush();
                boolean gotExitMessage = false;
                while (p.isAlive()) {
                    String in2 = reader.readLine();
                    System.out.println("Read " + in2);
                    if (in2 != null) {
                        System.out.printf("Read - %s\n", in2);
                        if (in2.contains("EndSession")) {
                            gotExitMessage = true;
                        }
                    } else {
                        Thread.sleep(100);
                    }
                }
                assertEquals(0, p.exitValue());
                assertTrue(gotExitMessage);
            });
        }
    }
}
```

A.4 SSTAF/src/applications/analyzer/src/test/java/analyzer/SSTAFAnalyzerTest.java

```
package analyzer;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.condition.EnabledOnOs;
import org.junit.jupiter.api.condition.OS;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;
public class SSTAFAnalyzerTest {
    /**
     * Unzip the zip file to the destination directory
     *
     * @param zipFilePath
     * @param destDir
     */
    private void unzip(Path zipFilePath, Path destDir) throws IOException {
        File dir = destDir.toFile();
        // create output directory if it doesn't exist
        if (!dir.exists()) {
            Files.createDirectories(destDir);
        }
        FileInputStream fis;
        //buffer for read and write data to file
        byte[] buffer = new byte[1024];
        try {
            fis = new FileInputStream(zipFilePath.toFile());
            ZipInputStream zis = new ZipInputStream(fis);
            ZipEntry ze = zis.getNextEntry();
            while (ze != null) {
                String fileName = ze.getName();
                Path entryPath = Path.of(destDir.toString(), fileName);
                System.out.println("Unzipping to " + entryPath.normalize().toAbsolutePath());
                if (ze.isDirectory()) {
                    Files.createDirectories(entryPath);
                } else {
                    File newFile = entryPath.toFile();
                    FileOutputStream fos = new FileOutputStream(newFile);
                    int len;
                    while ((len = zis.read(buffer)) > 0) {
                        fos.write(buffer, 0, len);
                    }
                    fos.close();
                }
                //close this ZipEntry
                zis.closeEntry();
                ze = zis.getNextEntry();
            }
            //close last ZipEntry
            zis.closeEntry();
            zis.close();
            fis.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    Path extractDistribution() throws IOException {
        String cwdString = System.getProperty("user.dir");
```

```
        Path analyzerZip = Path.of(cwdString,
                "build", "distributions", "sstaf-analyzer.zip");
        Path tmpDir = Path.of(cwdString, "build", "tmp");
        Path expandedDir = Files.createTempDirectory(tmpDir, "SSTAFAnalyzerTest");
        unzip(analyzerZip.normalize().toAbsolutePath(),
                expandedDir.normalize().toAbsolutePath());
        return Path.of(expandedDir.toString(), "sstaf-analyzer");
    }
    void runItLinux(Path installationDir) throws IOException {
        Path binDir = Path.of(installationDir.toString(), "bin");
        Path logDir = Path.of(installationDir.toString(), "logs");
        Files.createDirectory(logDir);
        File outFile = Path.of(logDir.toString(), "stdout").toFile();
        File errFile = Path.of(logDir.toString(), "stderr").toFile();
        ProcessBuilder builder = new ProcessBuilder();
        builder.redirectOutput(outFile);
        builder.redirectError(errFile);
        builder.directory(binDir.toFile())
                .command("bash", "sstaf-analyzer", "fred");
        builder.start();
    }
    @Test
    @EnabledOnOs(value = {OS.LINUX, OS.MAC})
    void doit() throws Exception {
        Path installationDir = extractDistribution();
        runItLinux(installationDir);
    }
}
```

A.5 SSTAF/src/build.gradle

```
//
// Necessary to resolve the dependencies for the SSTAF plugin
//
buildscript {
    repositories {
        mavenCentral()
        mavenLocal()
    }
    dependencies {
        classpath 'org.ow2.asm:asm:9.2'
        classpath 'org.projectlombok:lombok:1.18.22'
        classpath 'com.fasterxml.jackson.core:jackson-core:2.13.1'
        classpath 'com.fasterxml.jackson.core:jackson-databind:2.13.1'
        classpath 'com.fasterxml.jackson.core:jackson-annotations:2.13.1'
    }
}
plugins {
    id 'java-library'
    id 'jvm-test-suite'
    id 'maven-publish'
    id 'io.freefair.lombok' version '6.4.1'
    id 'sstaf' version '1.0'
}
repositories {
    mavenCentral()
    mavenLocal()
}
allprojects {
    sourceCompatibility = 11
    targetCompatibility = 11
}
/*
 * Many libraries have yet to be upgrades as fully-compliant modules. This block defines modules
for specific
 * legacy libraries. It works with the plugin in the buildSrc diqrectory,
 */
Set<Project> modules = subprojects.findAll { project ->
    project.path.contains('mil.') || project.path.contains('edu')
}

//
// Some libraries are not modules yet, work around it.
//
configure(modules) {
    apply plugin: 'java-library'
    apply plugin: 'jvm-test-suite'
    apply plugin: 'sstaf'
    apply plugin: 'io.freefair.lombok'
    apply plugin: 'maven-publish'
    version = '1.0.0-SNAPSHOT'
    String moduleName = getPath().split(":").last()
    int splitPoint = moduleName.lastIndexOf('.')
    String pomGroupId = moduleName.substring(0, splitPoint)
    String pomArtifactId = moduleName.substring(splitPoint + 1)
    repositories {
        mavenCentral()
        mavenLocal()
    }
    java {
        modularity.inferModulePath = true
        //
        // Cause the hashResources task to execute before jar. This enables the checksum files to
be
        // included in the jar.
```

```
        //
        tasks.named('jar') {
            it.dependsOn(tasks.named('hashResources'))
        }
        tasks.named('assemble') {
            it.dependsOn(tasks.named('integrationTest'))
        }
    }
    sstaf {
        // This does not have to be a complete description (e.g. here 'org.apache.commons.collect
ions' does not export anything here).
        // It only needs to be good enough to work in the context of this application we are buil
ding.
        module('commons-math3-3.6.1.jar', 'commons.math3', '3.6.1') {
            exports('org.apache.commons.math3.random')
            exports('org.apache.commons.math3.analysis.function')
            exports('org.apache.commons.math3.analysis.polynomials')
            exports('org.apache.commons.math3.analysis')
        }
        module('jopt-simple-4.6.jar', 'jopt.simple', '4.6')
        module('jmh-generator-annprocess-1.21.jar', 'jmh.generator.annprocess', '1.21')
        module('jmh-core-1.21.jar', 'jmh.core', '1.21') {
            exports('org.openjdk.jmh.annotations')
            exports('org.openjdk.jmh')
        }
        //
        // CSV support for ANSUR
        //
        module('commons-csv-1.8.jar', 'commons.csv', '1.8') {
            exports 'org.apache.commons.csv'
        }
        //
        // Set the hashing algorithm to be used for the resources.
        //
        hashAlgorithm = 'SHA-384'
    }
    dependencies {
        implementation group: 'org.slf4j', name: 'slf4j-api', version: '1.7.32'
        implementation group: 'org.apache.commons', name: 'commons-math3', version: '3.6.1'
        implementation 'com.fasterxml.jackson.core:jackson-core:2.13.1'
        implementation 'com.fasterxml.jackson.core:jackson-databind:2.13.1'
        implementation 'com.fasterxml.jackson.core:jackson-annotations:2.13.1'
        implementation 'org.projectlombok:lombok:1.18.22'
        testImplementation 'org.junit.jupiter:junit-jupiter:5.9.0'
        testRuntimeOnly 'org.junit.jupiter:junit-jupiter:5.9.0'
        testRuntimeOnly  'org.apiguardian:apiguardian-api:1.1.2'
        testRuntimeOnly group: 'org.slf4j', name: 'slf4j-simple', version: '1.7.32'
    }
    publishing {
        publications {
            maven(MavenPublication) {
               groupId = pomGroupId
               artifactId = pomArtifactId
                from components.java
            }
        }
        repositories {
            maven {
                name = 'DI2ENexus'
                //credentials(PasswordCredentials)
                url = "https://nexus.di2e.net/nexus3/repository/Private_SSTAF_Maven/"
            }
        }
    }
    test {
        useJUnitPlatform()
    }
    testing {
```

```
        suites {
            integrationTest(JvmTestSuite) {
                dependencies {
                    implementation project
                    implementation group: 'org.slf4j', name: 'slf4j-api', version: '1.7.32'
                    implementation group: 'org.apache.commons', name: 'commons-math3', version: '
3.6.1'
                    implementation 'com.fasterxml.jackson.core:jackson-core:2.13.1'
                    implementation 'com.fasterxml.jackson.core:jackson-databind:2.13.1'
                    implementation 'com.fasterxml.jackson.core:jackson-annotations:2.13.1'
                    implementation 'org.junit.jupiter:junit-jupiter:5.8.2'
                    implementation  'org.apiguardian:apiguardian-api:1.1.2'
                    runtimeOnly 'org.junit.jupiter:junit-jupiter:5.8.2'
                    runtimeOnly  'org.apiguardian:apiguardian-api:1.1.2'
                    runtimeOnly group: 'org.slf4j', name: 'slf4j-simple', version: '1.7.32'
                }
                targets {
                    all {
                        testTask.configure {
                            shouldRunAfter(test)
                        }
                    }
                }
            }
        }
    }
    tasks.named('check') {
        dependsOn(testing.suites.integrationTest)
    }
}
```

A.6 SSTAF/src/features/ansurHandler/mil.devcom_sc.ansur.api/build.gradle

```
dependencies {
    implementation project(':framework:mil.sstaf.core')
    implementation project(':features:ansurHandler:mil.devcom_sc.ansur.messages')
}
```

A.7 SSTAF/src/features/ansurHandler/mil.devcom_sc.ansur.api/src/main/java/mil/devcom_sc/ansur/api/ANSURConfiguration.java

```
package mil.devcom_sc.ansur.api;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import mil.devcom_sc.ansur.api.constraints.Constraint;
import mil.sstaf.core.features.FeatureConfiguration;
import java.util.List;
@SuperBuilder
@Jacksonized
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
@EqualsAndHashCode(callSuper = true)
public class ANSURConfiguration extends FeatureConfiguration {
    @Getter
    private List<Constraint> constraints;
}
```

A.8 SSTAF/src/features/ansurHandler/mil.devcom_sc.ansur.api/src/main/java/mil/devcom_sc/ansur/api/ANSURIIAnthropometry.java

```
package mil.devcom_sc.ansur.api;
import mil.devcom_sc.ansur.messages.Handedness;
import mil.devcom_sc.ansur.messages.Sex;
import mil.devcom_sc.ansur.messages.ValueKey;
import mil.sstaf.core.features.Handler;
import java.util.Optional;
public interface ANSURIIAnthropometry extends Handler {
    /**
     * Provides the {@code Sex} of the {@code Entity} to which this
     * anthropometry data is assigned.
     *
     * @return the Sex
     */
    Sex getSex();
    /**
     * Provides the height of the {@code Entity} in centimeters to which this
     * anthropometry data is assigned.
     *
     * @return the height in cm.
     */
    double getHeight_cm();
    /**
     * Provides the span of the {@code Entity} in centimeters.
     *
     * @return the height in cm.
     */
    double getSpan_cm();
    /**
     * Provides the weight of the {@code Entity} in kilograms.
     *
     * @return the weight in kg
     */
    double getWeight_kg();
    /**
     * Retrieves a value from the data for this instance.
     *
     * @param key the key for the value to retrieve
     * @return the value if it exists, null if not
     */
    Object getValue(ValueKey key);
    /**
     * Retrieves a value for this instance of anthropometry as a String.
     *
     * @param key the key for the value to retrieve
     * @return an {@code Optional} containing the value if it exists and
     * is a {@code String}. If the value exists but is not a {@code String} the
     * {@code Optional} will be empty.
     */
    Optional<String> getStringValue(ValueKey key);
    /**
     * Retrieves a value for this instance of anthropometry as a Double.
     *
     * @param key the key for the value to retrieve
     * @return an {@code Optional} containing the value if it exists and
     * is a {@code Double}. If the value exists but is not a {@code Double} the
     * {@code Optional} will be empty.
     */
    Optional<Double> getDoubleValue(ValueKey key);
    /**
     * Retrieves a value for this instance of anthropometry as a Integer.
```

```
     *
     * @param key the key for the value to retrieve
     * @return an {@code Optional} containing the value if it exists and
     * is a {@code Integer}. If the value exists but is not a {@code Integer} the
     * {@code Optional} will be empty.
     */
    Optional<Integer> getIntegerValue(ValueKey key);
    /**
     * Retrieves the Handedness (LEFT or RIGHT) of the subject
     * @return the Handedness
     */
    Handedness getHandedness();
}
```

A.9 SSTAF/src/features/ansurHandler/mil.devcom_sc.ansur.api/src/main/java/mil/devcom_sc/ansur/api/constraints/Constraint.java

```
package mil.devcom_sc.ansur.api.constraints;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import mil.devcom_sc.ansur.messages.ValueKey;
import java.util.Objects;
/**
 * Interface for classes that perform filtering constraints on
 * {@code CSVRecords}. These constraints can be used to select
 * ANSUR records that match criteria.
 */
@SuperBuilder
@Jacksonized
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
@EqualsAndHashCode
public class Constraint {
    @Getter
    protected String propertyName;
    protected Constraint(ConstraintBuilder<?, ?> builder) {
        Objects.requireNonNull(builder.propertyName, "propertyName");
        ValueKey.matchHeaderName(builder.propertyName);
        this.propertyName = builder.propertyName;
    }
}
```

A.10 SSTAF/src/features/ansurHandler/mil.devcom_sc.ansur.api/src/main/java/mil/devcom_sc/ansur/api/constraints/DoubleConstraint.java

```
package mil.devcom_sc.ansur.api.constraints;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
/**
 * An implementation of {@code Constraint} that matches against
 * a range of Doubles.
 */
@SuperBuilder
@Jacksonized
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
@EqualsAndHashCode(callSuper = true)
public class DoubleConstraint extends Constraint {
    @Getter
    private final double lowerBound;
    @Getter
    private final double upperBound;
    @Getter
    private final double equals;
}
```

A.11 SSTAF/src/features/ansurHandler/mil.devcom_sc.ansur.api/src/main/java/mil/devcom_sc/ansur/api/constraints/IntegerConstraint.java

```
package mil.devcom_sc.ansur.api.constraints;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
/**
 * An implementation of {@code Constraint} that matches against
 * a range of integers.
 */
@SuperBuilder
@Jacksonized
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
@EqualsAndHashCode(callSuper = true)
public class IntegerConstraint extends Constraint {
    @Getter
    private int lowerBound;
    @Getter
    private int upperBound;
    @Getter
    private final int equals;
}
```

A.12 SSTAF/src/features/ansurHandler/mil.devcom_sc.ansur.api/src/main/java/mil/devcom_sc/ansur/api/constraints/StringConstraint.java

```
package mil.devcom_sc.ansur.api.constraints;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import java.util.Objects;
/**
 * An implementation of {@code Constraint} that matches against
 * a range of Doubles.
 */
@SuperBuilder
@Jacksonized
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
@EqualsAndHashCode(callSuper = true)
public class StringConstraint extends Constraint {
    @Getter
    private final String matches;
    public StringConstraint(StringConstraintBuilder<?,?> builder) {
        super(builder);
        Objects.requireNonNull(builder.matches);
        this.matches = builder.matches;
    }
}
```

A.13 SSTAF/src/features/ansurHandler/mil.devcom_sc.ansur.api/src/main/java/module-info.java

```
module mil.devcom_sc.ansur.api {
    exports mil.devcom_sc.ansur.api;
    exports mil.devcom_sc.ansur.api.constraints;
    requires mil.sstaf.core;
    requires transitive  mil.devcom_sc.ansur.messages;
    opens mil.devcom_sc.ansur.api to com.fasterxml.jackson.databind;
    opens mil.devcom_sc.ansur.api.constraints to com.fasterxml.jackson.databind;
}
```

A.14 SSTAF/src/features/ansurHandler/mil.devcom_sc.ansur.api/src/test/java/mil/devcom_sc/ansur/api/constraints/DoubleConstraintTest.java

```
package mil.devcom_sc.ansur.api.constraints;
import mil.sstaf.core.json.JsonLoader;
import mil.sstaf.core.util.SSTAFException;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;
import java.nio.file.Path;
public class DoubleConstraintTest {

    @Test
    public void noPropertyNameThrows() {
        Assertions.assertThrows(NullPointerException.class, () -> {
            DoubleConstraint constraint = DoubleConstraint.builder()
                    .lowerBound(6)
                    .upperBound(13).build();
        });
    }

    @Test
    public void factoryWorks() {
        Assertions.assertDoesNotThrow(() -> {
            String json = "{ \"class\":\"mil.devcom_sc.ansur.api.constraints.DoubleConstraint\",\
"propertyName\":\"cervicaleheight\", \"lowerBound\":3.9, \"upperBound\":4.9}";
            DoubleConstraint constraint = new JsonLoader().load(json, DoubleConstraint.class, Pat
h.of("Bob"));
        });
    }
    @Test
    public void noPropertyNameThrows2() {
        Assertions.assertThrows(SSTAFException.class, () -> {
            String json = "{ \"class\":\"mil.devcom_sc.ansur.api.constraints.DoubleConstraint\",
\"lowerBound\":3.9, \"upperBound\":4.9}";
            DoubleConstraint constraint = new JsonLoader().load(json, DoubleConstraint.class, Pat
h.of("Bob"));
        });
    }
}
```

A.15 SSTAF/src/features/ansurHandler/mil.devcom_sc.ansur.api/src/test/java/mil/devcom_sc/ansur/api/constraints/IntegerConstraintTest.java

```
package mil.devcom_sc.ansur.api.constraints;
import mil.sstaf.core.json.JsonLoader;
import mil.sstaf.core.util.SSTAFException;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;
import java.nio.file.Path;
public class IntegerConstraintTest {
    @Test
    public void noPropertyNameThrows() {
        Assertions.assertThrows(NullPointerException.class, () -> {
            IntegerConstraint constraint = IntegerConstraint.builder()
                    .lowerBound(6)
                    .upperBound(13).build();
        });
    }

    @Test
    public void missingPropertyThrows() {
        Assertions.assertThrows(SSTAFException.class, () -> {
            String json = "{ \"class\":\"mil.devcom_sc.ansur.api.constraints.IntegerConstraint\",
\"lowerBound\":6, \"upperBound\":13}";
            IntegerConstraint constraint = new JsonLoader().load(json, IntegerConstraint.class, P
ath.of("bob"));
        });
    }
}
```

A.16 SSTAF/src/features/ansurHandler/mil.devcom_sc.ansur.api/src/test/java/mil/devcom_sc/ansur/api/constraints/StringConstraintTest.java

```
package mil.devcom_sc.ansur.api.constraints;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;
public class StringConstraintTest {

    @Test
    public void noPropertyNameThrows() {
        Assertions.assertThrows(NullPointerException.class, () -> {
            StringConstraint constraint = StringConstraint.builder()
                    .matches("banana").build();
        });
    }
}
```

A.17 SSTAF/src/features/ansurHandler/mil.devcom_sc.ansur.handler/build.gradle

```
dependencies {
    implementation project(':framework:mil.sstaf.core')
    implementation project(':features:ansurHandler:mil.devcom_sc.ansur.api')
    implementation project(':features:ansurHandler:mil.devcom_sc.ansur.messages')
    implementation group: 'org.apache.commons', name: 'commons-csv', version: '1.8'
}
task copyTestResources(type: Copy) {
    from "${projectDir}/src/test/resources"
    into "${buildDir}/classes/test"
}
processTestResources.dependsOn copyTestResources
task copyResources(type: Copy) {
    from "${projectDir}/src/main/resources"
    into "${buildDir}/classes/main"
}
processResources.dependsOn copyResources
```

A.18 SSTAF/src/features/ansurHandler/mil.devcom_sc.ansur.handler/out/test/resources/log4j2-test.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="INFO">
    <Appenders>
        <Console name="Console" target="SYSTEM_OUT">
            <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"/>
        </Console>
    </Appenders>
    <Loggers>
        <Root level="trace">
            <AppenderRef ref="Console"/>
        </Root>
    </Loggers>
</Configuration>
```

A.19 SSTAF/src/features/ansurHandler/mil.devcom_sc.ansur.handler/out/test/resources/multiple_constraints.json

```
{
  "constraints" : [
    {
      "propertyName" : "Gender",
      "matches" : "Male"
    },
    {
      "propertyName" : "balloffootlength",
      "equals" : 198
    },
    {
      "propertyName" : "footlength",
      "equals" : 270
    },
    {
      "propertyName" : "lowerthighcircumference",
      "lowerBound" : 390,
      "upperBound" : 395
    },
    {
      "propertyName" : "SubjectsBirthLocation",
      "matches" : ".*uck.*"
    }
  ]
}
```

A.20 SSTAF/src/features/ansurHandler/mil.devcom_sc.ansur.handler/out/test/resources/subjectID.json

```
{
  "subjectID" : 10173
}
```

A.21 SSTAF/src/features/ansurHandler/mil.devcom_sc.ansur.handler/out/test/resources/subjectID_as_constraint.json

```
{
  "constraints" : [
    {
      "propertyName" : "subjectid",
      "equals" : 10173
    }
  ]
}
```

A.22 SSTAF/src/features/ansurHandler/mil.devcom_sc.ansur.handler/src/main/java/mil/devcom_sc/ansur/handler/ANSURIIHandler.java

```
package mil.devcom_sc.ansur.handler;
import mil.devcom_sc.ansur.api.ANSURConfiguration;
import mil.devcom_sc.ansur.api.ANSURIIAnthropometry;
import mil.devcom_sc.ansur.api.constraints.Constraint;
import mil.devcom_sc.ansur.handler.filters.Filter;
import mil.devcom_sc.ansur.handler.filters.FilterFactory;
import mil.devcom_sc.ansur.messages.*;
import mil.sstaf.core.entity.Address;
import mil.sstaf.core.entity.Message;
import mil.sstaf.core.features.BaseHandler;
import mil.sstaf.core.features.FeatureConfiguration;
import mil.sstaf.core.features.HandlerContent;
import mil.sstaf.core.features.ProcessingResult;
import mil.sstaf.core.util.SSTAFException;
import org.apache.commons.math3.random.MersenneTwister;
import java.io.IOException;
import java.io.InputStream;
import java.util.*;

/**
 * Implementation of a SSTAF {@code Handler} that provides access to
 * the ANSUR II dataset.
 * <p>
 * The ANSUR II data is included in the module as two comma-separated value
 * files.
 */
public class ANSURIIHandler extends BaseHandler implements ANSURIIAnthropometry {
    /*
     * Keys
     */
    public static final String CK_SUBJECT_ID = "subjectID";
    public static final String CK_CONSTRAINTS = "constraints";
    public static final String FEATURE_NAME = "ANSUR Anthropometry";
    private Map<ValueKey, Object> subjectMap;
    private double height_cm = -1;
    private double span_cm = -1;
    private double weight_kg = -1;
    private Sex sex = null;
    private Handedness handedness = null;
    public ANSURIIHandler() {
        super(FEATURE_NAME, 1, 0, 0, true, "Provides ANSUR II data");
    }
    @Override
    public void configure(FeatureConfiguration configuration) {
        super.configure(configuration);
        if (configuration instanceof ANSURConfiguration) {
            ANSURConfiguration ansurConfiguration = (ANSURConfiguration) configuration;
            List<Constraint> constraints = ansurConfiguration.getConstraints();
            List<Filter> filters = FilterFactory.from(constraints);
            MersenneTwister rng = new MersenneTwister(configuration.getSeed());
            List<String> resourceNames = List.of("/ansur/ANSUR II MALE Public.csv",
                    "/ansur/ANSUR II FEMALE Public.csv");
            List<InputStream> streams = new ArrayList<>(resourceNames.size());
            for (String s : resourceNames) {
                InputStream is;
                try {
                    is = getClass().getModule().getResourceAsStream(s);
                } catch (IOException e) {
                    throw new SSTAFException(e);
                }
```

```
                streams.add(is);
            }
            subjectMap = SubjectSelector.select(streams, filters, rng.nextLong());
        }
    }
    /**
     * {@inheritDoc}
     */
    @Override
    public void init() {
        if (isInitialized()) return;
        super.init();
    }
    /**
     * Provides the Sex of the selected subject
     *
     * @return the sex
     */
    @Override
    public Sex getSex() {
        if (sex == null) {
            String s = (String) subjectMap.get(ValueKey.GENDER);
            sex = Sex.valueOf(s.toUpperCase());
        }
        return sex;
    }
    /**
     * Provides the height (stature) of the given subject in cm.
     *
     * @return the height.
     */
    @Override
    public double getHeight_cm() {
        if (height_cm < 0) {
            Integer height_mm = (Integer) subjectMap.get(ValueKey.STATURE);
            this.height_cm = height_mm / 10.0;
        }
        return height_cm;
    }
    /**
     * Provides the span of the given subject in cm.
     *
     * @return the span.
     */
    @Override
    public double getSpan_cm() {
        if (span_cm < 0) { // not initialized
            Integer span_mm = (Integer) subjectMap.get(ValueKey.SPAN);
            this.span_cm = span_mm / 10.0;
        }
        return span_cm;
    }
    /**
     * Provides the weight if the subject in kg.
     *
     * @return the weight
     */
    @Override
    public double getWeight_kg() {
        if (weight_kg < 0) {
            Integer weight_dg = (Integer) subjectMap.get(ValueKey.WEIGHT_KG);
            this.weight_kg = weight_dg / 10.0;
        }
        return weight_kg;
    }
    /**
     * Retrieves the Handedness (LEFT or RIGHT) of the subject
     *
```

```
     * @return the Handedness
     */
    @Override
    public Handedness getHandedness() {
        if (handedness == null) {
            Object o = subjectMap.get(ValueKey.WRITING_PREFERENCE);
            if (o instanceof String) {
                String s = (String) o;
                if (s.contains("Left")) {
                    handedness = Handedness.LEFT;
                }
            } else {
                handedness = Handedness.RIGHT; //default for now
            }
        }
        return handedness;
    }
    /**
     * {@inheritDoc}
     */
    @Override
    public List<Class<? extends HandlerContent>> contentHandled() {
        return List.of(GetValueMessage.class);
    }
    /**
     * {@inheritDoc}
     */
    @Override
    public ProcessingResult process(HandlerContent arg, long scheduledTime_ms, long currentTime_m
s, Address from,
                                    long sequence, Address respondTo) {
        System.err.println("WE'RE HERE - " + arg);
        if (arg instanceof GetValueMessage) {
            GetValueMessage msg = (GetValueMessage) arg;
            Objects.requireNonNull(msg.key);
            GetValueResponse.GetValueResponseBuilder<?,?> builder =
                    GetValueResponse.builder();
            ValueKey key = msg.key;
            builder.valueKey(key);
            if (key.getType().equals(String.class)) {
                System.err.println("String query");
                Optional<String> ov = getStringValue(key);
                ov.ifPresent(builder::stringValue);
            } else if (key.getType().equals(Double.class)) {
                System.err.println("Double query");
                Optional<Double> ov = getDoubleValue(key);
                ov.ifPresent(builder::doubleValue);
            } else if (key.getType().equals(Integer.class)) {
                System.err.println("Integer query");
                Optional<Integer> ov = getIntegerValue(key);
                ov.ifPresent(builder::intValue);
            }
            GetValueResponse response = builder.build();
            System.err.println("Response is " + response);
            Message out = buildNormalResponse(response, sequence, respondTo);
            return ProcessingResult.of(out);
        } else {
            throw new SSTAFException(arg.getClass().getName() + " is not supported");
        }
    }
    /**
     * Provides a value from the subject
     *
     * @param key the key for the value to retrieve
     * @return the value wrapped in an Optional.
     */
    @Override
    public Object getValue(ValueKey key) {
```

```
        return subjectMap.get(key);
    }
    /**
     * {@inheritDoc}
     */
    @Override
    public Optional<String> getStringValue(ValueKey key) {
        Object val = subjectMap.get(key);
        if (val instanceof String) return Optional.of((String) val);
        return Optional.empty();
    }
    /**
     * {@inheritDoc}
     */
    @Override
    public Optional<Double> getDoubleValue(ValueKey key) {
        Object val = subjectMap.get(key);
        if (val instanceof Double) return Optional.of((Double) val);
        return Optional.empty();
    }
    /**
     * {@inheritDoc}
     */
    @Override
    public Optional<Integer> getIntegerValue(ValueKey key) {
        Object val = subjectMap.get(key);
        if (val instanceof Integer) return Optional.of((Integer) val);
        return Optional.empty();
    }
}
```

A.23 SSTAF/src/features/ansurHandler/mil.devcom_sc.ansur.handler/src/main/java/mil/devcom_sc/ansur/handler/SubjectSelector.java

```
package mil.devcom_sc.ansur.handler;
import mil.devcom_sc.ansur.handler.filters.Filter;
import mil.devcom_sc.ansur.messages.ValueKey;
import mil.sstaf.core.util.SSTAFException;
import org.apache.commons.csv.CSVFormat;
import org.apache.commons.csv.CSVParser;
import org.apache.commons.csv.CSVRecord;
import org.apache.commons.math3.random.MersenneTwister;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.Reader;
import java.util.*;
/**
 * Utility class for selecting a single ANSUR subject given a list
 * of {@code Contraint}s
 */
public class SubjectSelector {
    /**
     * Selects a single subject from all ANSUR records.
     * <p>
     * ANSUR records are filtered using the list of {@code BaseFilter} objects. If multiple match
es
     * are found, one is selected randomly.
     *
     * @param streams    the {@code InputStream}s from which to read the data
     * @param filters    the constraints to apply
     * @param randomSeed the seed for the random number generator
     * @return a {@code Map} that contains all of the fields for the subject
     */
    static Map<ValueKey, Object> select(final List<InputStream> streams,
                                        final List<Filter> filters,
                                        final long randomSeed) {
        List<CSVRecord> matches = findAllMatches(streams, filters);
        CSVRecord chosen = getChosen(matches, randomSeed);
        return parseSubject(chosen);
    }
    /**
     * Finds all {@code CSVRecord}s that match the criteria across multiple {@code InputStream}s.
     *
     * @param streams the InputStreams from which to read.
     * @param filters the {@code BaseFilter}s to apply
     * @return a List of CSVRecords that match the constraints.
     */
    static List<CSVRecord> findAllMatches(List<InputStream> streams, List<Filter> filters) {
        List<CSVRecord> matches = new ArrayList<>();
        for (InputStream stream : streams) {
            Objects.requireNonNull(stream, "stream");
            List<CSVRecord> csvRecords = findMatches(stream, filters);
            matches.addAll(csvRecords);
        }
        if (matches.isEmpty()) {
            StringBuilder sb = new StringBuilder();
            for (Filter f : filters) {
                sb.append("[").append(f).append("] ");
            }
            throw new SSTAFException("No subjects matched constraints of " + sb);
        }
        return matches;
    }
```

```
    /**
     * Selects one subject record from a list of records that have met the selection criteria
     *
     * @param matches    the list of subject records from which to choose.
     * @param randomSeed the seed for initializing the random number generator
     * @return the selected subject record
     */
    private static CSVRecord getChosen(final List<CSVRecord> matches, long randomSeed) {
        CSVRecord chosen;
        int numPossibilities = matches.size();
        if (numPossibilities == 0) {
            throw new SSTAFException("No subjects matched constraints");
        } else if (numPossibilities == 1) {
            chosen = matches.get(0);
        } else {
            MersenneTwister rng = new MersenneTwister(randomSeed);
            int selection = rng.nextInt(numPossibilities);
            chosen = matches.get(selection);
        }
        return chosen;
    }
    /**
     * Parses a subject record into a Map.
     *
     * @param chosen the record to parse
     * @return a Map full of ANSUR II metrics
     */
    static Map<ValueKey, Object> parseSubject(final CSVRecord chosen) {
        Map<ValueKey, Object> subjectMap = new EnumMap<>(ValueKey.class);
        for (ValueKey key : ValueKey.values()) {
            String valString = chosen.get(key.getHeaderLabel());
            if (key.getType() == Integer.class) {
                subjectMap.put(key, Integer.parseInt(valString));
            } else if (key.getType() == Double.class) {
                //
                // the ANSUR II database doesn't use double values yet.
                //
                subjectMap.put(key, Double.parseDouble(valString));
            } else {
                subjectMap.put(key, valString);
            }
        }
        return Collections.unmodifiableMap(subjectMap);
    }
    /**
     * Scans an {@code InputStream} for {@code CSVRecord}s that match the given {@code BaseFilter
}s.
     *
     * @param inputStream the InputStream for the CSV data source
     * @param filters     the Constraints to use to filter the records
     * @return a List of matching CSVRecords
     */
    static List<CSVRecord> findMatches(final InputStream inputStream, final List<Filter> filters)
{
        Objects.requireNonNull(inputStream, "inputStream ");
        List<CSVRecord> matches = new ArrayList<>();
        CSVRecord dup = null;
        try {
            Reader reader = new InputStreamReader(inputStream);
            CSVParser parser = new CSVParser(reader,
                    CSVFormat.DEFAULT.withFirstRecordAsHeader().withIgnoreSurroundingSpaces().wit
hIgnoreHeaderCase());
            for (final CSVRecord record: parser) {
                boolean itMatches = true;
                dup = record;
                for (final Filter filter : filters) {
                    itMatches = filter.matches(record);
                    if (!itMatches) break;
```

```
                }
                if (itMatches) {
                    matches.add(record);
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
            throw new SSTAFException("Failed to read " + dup, e);
        }
        return matches;
    }
}
```

A.24 SSTAF/src/features/ansurHandler/mil.devcom_sc.ansur.handler/src/main/java/mil/devcom_sc/ansur/handler/filters/BaseFilter.java

```
package mil.devcom_sc.ansur.handler.filters;
import lombok.Getter;
import mil.devcom_sc.ansur.messages.ValueKey;
import java.util.Objects;
/**
 * Interface for classes that perform filtering constraints on
 * {@code CSVRecords}. These constraints can be used to select
 * ANSUR records that match criteria.
 */
public abstract class BaseFilter implements Filter {
    @Getter
    private final ValueKey property;
    protected BaseFilter(mil.devcom_sc.ansur.api.constraints.Constraint constraint) {
        Objects.requireNonNull(constraint, "constraint");
        this.property = ValueKey.matchHeaderName(constraint.getPropertyName());
    }
}
```

A.25 SSTAF/src/features/ansurHandler/mil.devcom_sc.ansur.handler/src/main/java/mil/devcom_sc/ansur/handler/filters/DoubleFilter.java

```
package mil.devcom_sc.ansur.handler.filters;
import lombok.Getter;
import lombok.ToString;
import mil.devcom_sc.ansur.api.constraints.DoubleConstraint;
import org.apache.commons.csv.CSVRecord;
/**
 * An implementation of {@code BaseFilter} that matches against
 * a range of Doubles.
 */
@ToString
public class DoubleFilter extends BaseFilter {
    @Getter
    private final double lowerBound;
    @Getter
    private final double upperBound;
    @Getter
    private final double equals;
    private final boolean useEquals;
    public DoubleFilter(DoubleConstraint constraint) {
        super(constraint);
        if (constraint.getEquals() != 0) {
            this.equals = constraint.getEquals();
            this.useEquals = true;
        } else if (constraint.getLowerBound() == constraint.getUpperBound()) {
            this.equals = constraint.getUpperBound();
            this.useEquals = true;
        } else {
            this.equals = constraint.getEquals();
            this.useEquals = false;
        }
        double lowerBound;
        double upperBound;
        if (constraint.getUpperBound() > constraint.getLowerBound()) {
            upperBound = constraint.getUpperBound();
            lowerBound = constraint.getLowerBound();
        } else if (constraint.getUpperBound() == 0) {
            upperBound = Double.MAX_VALUE;
            lowerBound = constraint.getLowerBound();
        } else if (constraint.getLowerBound() == 0) {
            upperBound = constraint.getUpperBound();
            lowerBound = Double.MIN_VALUE;
        } else {
            upperBound = constraint.getLowerBound();
            lowerBound = constraint.getUpperBound();
        }
        this.upperBound = upperBound;
        this.lowerBound = lowerBound;
    }
    /**
     * {@inheritDoc}
     */
    @Override
    public boolean matches(CSVRecord record) {
        String valueString = record.get(getProperty().getHeaderLabel());
        double val = Double.parseDouble(valueString);
        if (useEquals) {
            return val == equals;
        } else {
            return val >= lowerBound && val <= upperBound;
        }
```

```
    }
}
```

A.26 SSTAF/src/features/ansurHandler/mil.devcom_sc.ansur.handler/src/main/java/mil/devcom_sc/ansur/handler/filters/Filter.java

```
package mil.devcom_sc.ansur.handler.filters;
import org.apache.commons.csv.CSVRecord;
public interface Filter {
    boolean matches(CSVRecord record) ;
}
```

A.27 SSTAF/src/features/ansurHandler/mil.devcom_sc.ansur.handler/src/main/java/mil/devcom_sc/ansur/handler/filters/FilterFactory.java

```
package mil.devcom_sc.ansur.handler.filters;
import mil.devcom_sc.ansur.api.constraints.Constraint;
import mil.devcom_sc.ansur.api.constraints.DoubleConstraint;
import mil.devcom_sc.ansur.api.constraints.IntegerConstraint;
import mil.devcom_sc.ansur.api.constraints.StringConstraint;
import mil.sstaf.core.util.SSTAFException;
import java.util.ArrayList;
import java.util.List;
public class FilterFactory {
    public static Filter from(Constraint c) {
        Filter filter;
        if (c instanceof DoubleConstraint) {
            filter = new DoubleFilter((DoubleConstraint) c);
        } else if (c instanceof IntegerConstraint) {
            filter = new IntegerFilter((IntegerConstraint) c);
        } else if (c instanceof StringConstraint) {
            filter = new StringFilter((StringConstraint) c);
        } else {
            throw new SSTAFException("Huh?");
        }
        return filter;
    }
    public static List<Filter> from(Iterable<Constraint> constraints) {
        List<Filter> filters = new ArrayList<>();
        for(var c : constraints) {
           filters.add(from(c));
        }
        return filters;
    }
}
```

A.28 SSTAF/src/features/ansurHandler/mil.devcom_sc.ansur.handler/src/main/java/mil/devcom_sc/ansur/handler/filters/IntegerFilter.java

```
package mil.devcom_sc.ansur.handler.filters;
import lombok.Getter;
import lombok.ToString;
import mil.devcom_sc.ansur.api.constraints.IntegerConstraint;
import org.apache.commons.csv.CSVRecord;
/**
 * An implementation of {@code BaseFilter} that matches against
 * a range of integers.
 */
@ToString
public class IntegerFilter extends BaseFilter {
    @Getter
    private final int lowerBound;
    @Getter
    private final int upperBound;
    @Getter
    private final int equals;
    private final boolean useEquals;
    public IntegerFilter(IntegerConstraint constraint) {
        super(constraint);
        if (constraint.getEquals() != 0) {
            this.equals = constraint.getEquals();
            this.useEquals = true;
        } else if (constraint.getLowerBound() == constraint.getUpperBound()) {
            this.equals = constraint.getUpperBound();
            this.useEquals = true;
        } else {
            this.equals = constraint.getEquals();
            this.useEquals = false;
        }
        int lowerBound;
        int upperBound;
        if (constraint.getUpperBound() > constraint.getLowerBound()) {
            upperBound = constraint.getUpperBound();
            lowerBound = constraint.getLowerBound();
        } else if (constraint.getUpperBound() == 0) {
            upperBound = Integer.MAX_VALUE;
            lowerBound = constraint.getLowerBound();
        } else if (constraint.getLowerBound() == 0) {
            upperBound = constraint.getUpperBound();
            lowerBound = Integer.MIN_VALUE;
        } else {
            upperBound = constraint.getLowerBound();
            lowerBound = constraint.getUpperBound();
        }
        this.upperBound = upperBound;
        this.lowerBound = lowerBound;
    }
    /**
     * {@inheritDoc}
     */
    @Override
    public boolean matches(CSVRecord record) {
        String valueString = record.get(getProperty().getHeaderLabel());
        int val = Integer.parseInt(valueString);
        if (useEquals) {
            return val == equals;
        } else {
            return val >= lowerBound && val <= upperBound;
        }
```

```
    }
}
```

A.29 SSTAF/src/features/ansurHandler/mil.devcom_sc.ansur.handler/src/main/java/mil/devcom_sc/ansur/handler/filters/StringFilter.java

```
package mil.devcom_sc.ansur.handler.filters;
import lombok.Getter;
import lombok.ToString;
import mil.devcom_sc.ansur.api.constraints.StringConstraint;
import org.apache.commons.csv.CSVRecord;
import java.util.Objects;
/**
 * An implementation of {@code BaseFilter} that matches against
 * a range of Doubles.
 */
@ToString
public class StringFilter extends BaseFilter {
    @Getter
    private final String matches;
    public StringFilter(StringConstraint constraint) {
        super(constraint);
        Objects.requireNonNull(constraint.getMatches());
        this.matches = constraint.getMatches();
    }
    /**
     * {@inheritDoc}
     */
    @Override
    public boolean matches(CSVRecord record) {
        String valueString = record.get(getProperty().getHeaderLabel());
        return valueString.matches(matches);
    }
}
```

A.30 SSTAF/src/features/ansurHandler/mil.devcom_sc.ansur.handler/src/main/java/module-info.java

```
import mil.devcom_sc.ansur.api.ANSURIIAnthropometry;
import mil.devcom_sc.ansur.handler.ANSURIIHandler;
import mil.sstaf.core.features.Feature;
import mil.sstaf.core.features.Handler;
module mil.devcom_sc.ansur.handler {
    exports mil.devcom_sc.ansur.handler;
    requires mil.sstaf.core;

    requires mil.devcom_sc.ansur.api;
    requires commons.csv;
    provides Feature with ANSURIIHandler;
    provides Handler with ANSURIIHandler;
    provides ANSURIIAnthropometry with ANSURIIHandler;
    opens mil.devcom_sc.ansur.handler to mil.sstaf.core;
}
```

A.31 SSTAF/src/features/ansurHandler/mil.devcom_sc.ansur.handler/src/test/java/mil/devcom_sc/ansur/handler/ANSURHandlerTest.java

```
package mil.devcom_sc.ansur.handler;
import mil.devcom_sc.ansur.api.ANSURConfiguration;
import mil.devcom_sc.ansur.api.ANSURIIAnthropometry;
import mil.devcom_sc.ansur.api.constraints.Constraint;
import mil.devcom_sc.ansur.api.constraints.IntegerConstraint;
import mil.devcom_sc.ansur.messages.GetValueMessage;
import mil.devcom_sc.ansur.messages.GetValueResponse;
import mil.devcom_sc.ansur.messages.Sex;
import mil.devcom_sc.ansur.messages.ValueKey;
import mil.sstaf.core.entity.Address;
import mil.sstaf.core.entity.EntityHandle;
import mil.sstaf.core.entity.Message;
import mil.sstaf.core.features.HandlerContent;
import mil.sstaf.core.features.Loaders;
import mil.sstaf.core.features.ProcessingResult;
import mil.sstaf.core.json.JsonLoader;
import mil.sstaf.core.util.Injector;
import mil.sstaf.core.util.SSTAFException;
import org.junit.jupiter.api.*;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.EnumSource;

import java.io.File;
import java.nio.file.Path;
import java.util.List;
import java.util.Optional;

import static mil.devcom_sc.ansur.messages.ValueKey.ACROMIAL_HEIGHT;
import static mil.devcom_sc.ansur.messages.ValueKey.SUBJECT_ID;
import static org.junit.jupiter.api.Assertions.*;

public class ANSURHandlerTest {
    public static String basepath = "src" + "/" + "test" + "/" + "resources" + "/";
    static ANSURConfiguration multiConstraintConfig;
    ANSURIIAnthropometry handler;

    @BeforeAll
    static void beginning() {
        multiConstraintConfig = new JsonLoader().load(Path.of(basepath, "multiple_constraints.jso
n"), ANSURConfiguration.class);
    }


    @BeforeEach
    void setUp() {
        System.out.println(new File(".").getName());
        try {
            System.setProperty("sstaf.preloadFeatureClasses", "true");
            Loaders.preloadFeatureClasses("mil.devcom_sc.ansur.handler.ANSURIIHandler");
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
            fail();
        }

        Optional<ANSURIIAnthropometry> optAgent = Loaders.load(ANSURIIAnthropometry.class, "ANSUR
Anthropometry", 0, 9);
        if (optAgent.isPresent()) {
            handler = optAgent.get();
        } else {
            Assertions.fail();
```

```
        }

        EntityHandle owner = EntityHandle.makeDummyHandle();
        Injector.inject(handler, owner);
    }


    @Nested
    @DisplayName("Tests that check normal operation")
    class HappyTests {

        @Test
        @DisplayName("Confirm that the ANSURHandler service can be loaded")
        void simpleTest() {
            assertNotNull(handler);
        }

        @Test
        @DisplayName("Confirm that multiple constraints can be applied")
        void multipleConstraintsWorks() {
            assertNotNull(handler);

            ANSURConfiguration ansurConfiguration = new JsonLoader().load(Path.of(basepath, "mult
iple_constraints.json"), ANSURConfiguration.class);

            assertDoesNotThrow(() -> {
                handler.configure(ansurConfiguration);
                handler.init();
                Optional<Integer> optVal = handler.getIntegerValue(SUBJECT_ID);
                if (optVal.isPresent()) {
                    Assertions.assertEquals(10303, optVal.get());
                } else {
                    Assertions.fail("Value for " + SUBJECT_ID.getHeaderLabel() + " was not found"
);
                }
            });
        }

        @Test
        @DisplayName("Confirm that an explicit SubjectID specification worksN")
        void subjectIDFromFileWorks() {
            assertNotNull(handler);

            assertDoesNotThrow(() -> {
                IntegerConstraint constraint = (IntegerConstraint) new JsonLoader().load(Path.of(
basepath, "subjectID.json"));
                ANSURConfiguration config = ANSURConfiguration.builder().constraints(List.of(cons
traint)).build();
                assertNotNull(config);
                config.setSeed(1342345L);
                handler.configure(config);
                handler.init();
                Optional<Integer> optVal = handler.getIntegerValue(ACROMIAL_HEIGHT);
                if (optVal.isPresent()) {
                    Assertions.assertEquals(1335, optVal.get());
                } else {
                    Assertions.fail("Value for " + ACROMIAL_HEIGHT.getHeaderLabel() + " was not f
ound");
                }
            });
        }


        @Test
        @DisplayName("Confirm that SubjectID formulated as a constraint works")
        void subjectIDAsConstraintWorks() {
            assertNotNull(handler);
```

```
            assertDoesNotThrow(() -> {
                IntegerConstraint constraint = (IntegerConstraint) new JsonLoader().load(Path.of(
basepath, "subjectID_as_constraint.json"));
                ANSURConfiguration config = ANSURConfiguration.builder().constraints(List.of(cons
traint)).build();
                config.setSeed(1234234L);
                handler.configure(config);
                handler.init();
                Optional<Integer> optVal = handler.getIntegerValue(ACROMIAL_HEIGHT);
                if (optVal.isPresent()) {
                    Assertions.assertEquals(1335, optVal.get());
                } else {
                    Assertions.fail("Value for " + ACROMIAL_HEIGHT.getHeaderLabel() + " was not f
ound");
                }
            });
        }


        @Test
        void subjectIDWorks() {
            assertNotNull(handler);

            assertDoesNotThrow(() -> {
                List<Constraint> constraints = List.of(IntegerConstraint.builder().propertyName(S
UBJECT_ID.getHeaderLabel()).equals(10173).build());
                ANSURConfiguration configuration = ANSURConfiguration.builder().constraints(const
raints).build();
                configuration.setSeed(12345L);
                handler.configure(configuration);
                handler.init();
                Optional<Integer> optVal = handler.getIntegerValue(ACROMIAL_HEIGHT);
                if (optVal.isPresent()) {
                    Assertions.assertEquals(1335, optVal.get());
                } else {
                    Assertions.fail("Value for " + ACROMIAL_HEIGHT.getHeaderLabel() + " was not f
ound");
                }
            });

        }

        @Test
        @DisplayName("Confirm that at least one Female < 65kg can be found")
        void findLightFemales() {
            assertNotNull(handler);
            ANSURConfiguration ansurConfiguration = new JsonLoader().load(Path.of(basepath, "Fema
lesUnder65Kg.json"), ANSURConfiguration.class);

            assertDoesNotThrow(() -> {
                handler.configure(ansurConfiguration);
                handler.init();
                Optional<Integer> optVal = handler.getIntegerValue(SUBJECT_ID);
                Assertions.assertTrue(optVal.isPresent());
            });
        }

        @Test
        @DisplayName("Confirm that message handling works")
        void messageHandlingWorks() {
            assertNotNull(handler);

            ANSURConfiguration ansurConfiguration = new JsonLoader().load(Path.of(basepath, "mult
iple_constraints.json"), ANSURConfiguration.class);

            assertDoesNotThrow(() -> {
                handler.configure(ansurConfiguration);
                handler.init();
```

```
                GetValueMessage msg = GetValueMessage.of(ValueKey.BICEPS_CIRCUMFERENCE_FLEXED);
                Assertions.assertEquals(1, handler.contentHandled().size());
                Assertions.assertEquals(GetValueMessage.class, handler.contentHandled().get(0));

                ProcessingResult pr = handler.process(msg, 0, 0, Address.NOWHERE, 12345, Address.
NOWHERE);
                Assertions.assertEquals(1, pr.messages.size());
                Message response = pr.messages.get(0);
                Object content = response.getContent();
                Assertions.assertTrue(content instanceof GetValueResponse);
                GetValueResponse gvr = (GetValueResponse) content;

                Optional<Integer> optVal = gvr.getIntegerValue();
                if (optVal.isPresent()) {
                    Assertions.assertEquals(341, optVal.get());
                } else {
                    Assertions.fail("Value for " + ValueKey.BICEPS_CIRCUMFERENCE_FLEXED.getHeader
Label() + " was not found");
                }
            });
        }

        @DisplayName("Confirm that every ValueKey can be accessed")
        @ParameterizedTest(name = "[{index}] -- {0}")
        @EnumSource(ValueKey.class)
        void allValuesCanBeAccessed(ValueKey key) {
            assertNotNull(handler);

            assertDoesNotThrow(() -> {
                handler.configure(multiConstraintConfig);
                handler.init();

                Assertions.assertTrue(handler.getHeight_cm() > 0.0);
                assertEquals(handler.getSex(), Sex.MALE);
                Assertions.assertNotNull(handler.getName());
                Assertions.assertTrue(handler.getSpan_cm() > 0.0);
                Assertions.assertTrue(handler.getWeight_kg() > 0.0);

                GetValueMessage msg = GetValueMessage.of(key);
                Assertions.assertEquals(1, handler.contentHandled().size());
                Assertions.assertEquals(GetValueMessage.class, handler.contentHandled().get(0));

                ProcessingResult pr = handler.process(msg, 0, 0, Address.NOWHERE, 12345, Address.
NOWHERE);
                Assertions.assertEquals(1, pr.messages.size());
                Message response = pr.messages.get(0);
                Object content = response.getContent();
                Assertions.assertTrue(content instanceof GetValueResponse);
                GetValueResponse gvr = (GetValueResponse) content;

                if (key.getType() == Integer.class) {
                    Optional<Integer> optVal1 = gvr.getIntegerValue();
                    Optional<Integer> optVal2 = handler.getIntegerValue(key);
                    Optional<String> optVal3 = handler.getStringValue(key);
                    if (optVal1.isEmpty() || optVal2.isEmpty()) {
                        Assertions.fail("Integer value for " + key.getHeaderLabel() + " was not f
ound");
                    }
                    Assertions.assertEquals(optVal1.get(), optVal2.get());
                    Assertions.assertTrue(optVal3.isEmpty());
                } else if (key.getType() == String.class) {
                    Optional<String> optVal1 = gvr.getStringValue();
                    Optional<String> optVal2 = handler.getStringValue(key);
                    Optional<Integer> optVal3 = handler.getIntegerValue(key);
                    if (optVal1.isEmpty() || optVal2.isEmpty()) {
                        Assertions.fail("String value for " + key.getHeaderLabel() + " was not fo
und");
```

```
                    }
                    Assertions.assertEquals(optVal1.get(), optVal2.get());
                    Assertions.assertTrue(optVal3.isEmpty());
                } else {
                    Assertions.fail(key.name() + " has bad type");
                }

                //
                // There are no double values, so this should always return an empty optional
                //
                Optional<Double> optionalDouble = handler.getDoubleValue(key);
                Assertions.assertTrue(optionalDouble.isEmpty());
            });
        }

    }

    @Nested
    @DisplayName("Test failure modes")
    class FailureTests {

        @Test
        @DisplayName("Confirm that if an unsupported HandleContent type is supplied then an excep
tion is thrown")
        void badMessageThrows() {
            assertNotNull(handler);
            assertDoesNotThrow(() -> {
                List<Constraint> constraints = List.of(IntegerConstraint.builder().propertyName(S
UBJECT_ID.getHeaderLabel()).equals(10173).build());
                ANSURConfiguration configuration = ANSURConfiguration.builder().constraints(const
raints).build();
                configuration.setSeed(12345L);
                handler.configure(configuration);
                handler.init();
            });
            assertThrows(SSTAFException.class, () -> handler.process(HandlerContent.builder().bui
ld(), 0, 0, Address.NOWHERE, 0, Address.NOWHERE));
        }


    }
}
```

A.32 SSTAF/src/features/ansurHandler/mil.devcom_sc.ansur.handler/src/test/java/mil/devcom_sc/ansur/handler/SubjectSelectorTest.java

```
package mil.devcom_sc.ansur.handler;
import mil.devcom_sc.ansur.api.constraints.Constraint;
import mil.devcom_sc.ansur.api.constraints.IntegerConstraint;
import mil.devcom_sc.ansur.api.constraints.StringConstraint;
import mil.devcom_sc.ansur.handler.filters.Filter;
import mil.devcom_sc.ansur.handler.filters.FilterFactory;
import mil.devcom_sc.ansur.handler.filters.IntegerFilter;
import mil.devcom_sc.ansur.handler.filters.StringFilter;
import mil.devcom_sc.ansur.messages.ValueKey;
import mil.sstaf.core.util.SSTAFException;
import org.apache.commons.csv.CSVRecord;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;
public class SubjectSelectorTest {
    @Test
    public void matchSubjectWorks() {
        Assertions.assertDoesNotThrow(() -> {
            List<InputStream> streams = getStreamList();

            IntegerConstraint constraint = IntegerConstraint.builder()
                    .propertyName(ValueKey.SUBJECT_ID.getHeaderLabel())
                    .upperBound(10127).lowerBound(10127).build();
            IntegerFilter filter = new IntegerFilter(constraint);
            List<Filter> filters = List.of(filter);
            Map<ValueKey, Object> subject = SubjectSelector.select(streams, filters, 1234);
            Assertions.assertNotNull(subject);
            Assertions.assertEquals(1371, subject.get(ValueKey.ACROMIAL_HEIGHT));
        });
    }
    @Test
    public void noMatchThrows() {
        Assertions.assertThrows(SSTAFException.class, () -> {
            List<InputStream> streams = getStreamList();
            IntegerConstraint constraint = IntegerConstraint.builder().propertyName(ValueKey.WAIS
T_BREADTH.getHeaderLabel())
                    .lowerBound(123456).build();
            IntegerFilter filter = new IntegerFilter(constraint);
            List<Filter> filters = List.of(filter);
            Map<ValueKey, Object> subject = SubjectSelector.select(streams, filters, 1234);
        });
    }
    @Test
    public void anyMaleGivesSomething() {
        Assertions.assertDoesNotThrow(() -> {
            List<InputStream> streams = getStreamList();
            StringConstraint constraint = StringConstraint.builder().propertyName(ValueKey.GENDER
.getHeaderLabel())
                    .matches("Male").build();
            StringFilter filter = new StringFilter(constraint);
            List<Filter> filters = List.of(filter);
            Map<ValueKey, Object> subject = SubjectSelector.select(streams, filters, 1234);
            Assertions.assertNotNull(subject);
            Assertions.assertEquals("Male", subject.get(ValueKey.GENDER));
            Assertions.assertNotEquals("Female", subject.get(ValueKey.GENDER));
```

```
        });
    }
    @Test
    public void anyFemaleGivesSomething() {
        Assertions.assertDoesNotThrow(() -> {
            List<InputStream> streams = getStreamList();
            StringConstraint constraint = StringConstraint.builder().propertyName(ValueKey.GENDER
.getHeaderLabel())
                    .matches("Female").build();
            StringFilter filter = new StringFilter(constraint);
            List<Filter> filters = List.of(filter);
            Map<ValueKey, Object> subject = SubjectSelector.select(streams, filters, 1234);
            Assertions.assertNotNull(subject);
            Assertions.assertEquals("Female", subject.get(ValueKey.GENDER));
            Assertions.assertNotEquals("Male", subject.get(ValueKey.GENDER));
        });
    }
    @Test
    public void noFiltersGivesSomething() {
        Assertions.assertDoesNotThrow(() -> {
            List<InputStream> streams = getStreamList();
            List<Filter> filters = List.of();
            Map<ValueKey, Object> subject = SubjectSelector.select(streams, filters, 1234);
            Assertions.assertNotNull(subject);
            Integer value = (Integer) subject.get(ValueKey.CHEST_CIRCUMFERENCE);
            Assertions.assertTrue(value > 10);
        });
    }
    @Test
    public void all11BsAreFound() {
        Assertions.assertDoesNotThrow(() -> {
            StringConstraint contraint = StringConstraint.builder().propertyName(ValueKey.PRIMARY
_MOS.getHeaderLabel())
                    .matches("11B").build();
            StringFilter filter= new StringFilter(contraint);
            List<Filter> filters = List.of(filter);
            List<CSVRecord> subjects = SubjectSelector.findAllMatches(getStreamList(), filters);
            Assertions.assertNotNull(subjects);
            Assertions.assertEquals(671, subjects.size());
        });
    }

    @Test
    public void multipleFiltersWork() {
        Assertions.assertDoesNotThrow(() -> {
            List<InputStream> streams = getStreamList();
            List<Constraint> constraints = List.of(
                    StringConstraint.builder().propertyName(ValueKey.GENDER.getHeaderLabel())
                            .matches("Male").build(),
                    IntegerConstraint.builder().propertyName(ValueKey.BALL_OF_FOOT_LENGTH.getHead
erLabel()).
                            lowerBound(198).upperBound(198).build(),
                    IntegerConstraint.builder().propertyName(ValueKey.FOOT_LENGTH.getHeaderLabel(
)).
                            lowerBound(270).upperBound(270).build(),
                    IntegerConstraint.builder().propertyName(ValueKey.LOWER_THIGH_CIRCUMFERENCE.g
etHeaderLabel()).
                            lowerBound(390).upperBound(395).build(),
                    StringConstraint.builder().propertyName(ValueKey.SUBJECTS_BIRTH_LOCATION.getH
eaderLabel())
                            .matches(".*uck.*").build()  // Kentucky!
            );
            List<Filter> filters = FilterFactory.from(constraints);
            Map<ValueKey, Object> subject = SubjectSelector.select(streams, filters, 1234);
            Assertions.assertNotNull(subject);
            //
            // It should be this guy.
            //
```

```
            Assertions.assertEquals(10303, subject.get(ValueKey.SUBJECT_ID));
        });
    }
    private List<InputStream> getStreamList() {
        List<String> resources = List.of("src/main/resources/ansur/ANSUR II MALE Public.csv", "sr
c/main/resources/ansur/ANSUR II FEMALE Public.csv");
        return resources.stream().map(this::makeStream).collect(Collectors.toList());
    }
    private InputStream makeStream(String fileName) {
        try {
            return new FileInputStream(fileName);
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
        return null;
    }
}
```

A.33 SSTAF/src/features/ansurHandler/mil.devcom_sc.ansur.handler/src/test/java/mil/devcom_sc/ansur/handler/filters/DoubleFilterTest.java

```
package mil.devcom_sc.ansur.handler.filters;
import mil.devcom_sc.ansur.api.constraints.DoubleConstraint;
import mil.devcom_sc.ansur.messages.ValueKey;
import mil.sstaf.core.json.JsonLoader;
import mil.sstaf.core.util.SSTAFException;
import org.apache.commons.csv.CSVFormat;
import org.apache.commons.csv.CSVParser;
import org.apache.commons.csv.CSVRecord;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;
import java.io.IOException;
import java.io.Reader;
import java.io.StringReader;
import java.nio.file.Path;
import java.util.List;
import static mil.devcom_sc.ansur.api.constraints.DoubleConstraint.builder;
public class DoubleFilterTest {

    String value1 = ValueKey.EAR_LENGTH.getHeaderLabel();
    String value2 = ValueKey.FOOT_LENGTH.getHeaderLabel();
    String value3 = ValueKey.SUBJECTS_BIRTH_LOCATION.getHeaderLabel();
    String value4 = ValueKey.WEIGHT_LBS.getHeaderLabel();
    String value5 = ValueKey.WAIST_DEPTH.getHeaderLabel();
    CSVParser getParser() throws IOException {
        String fakeCSV
                = String.format("%s,%s,%s,%s,%s\n",
                value1, value2, value3, value4, value5) +
                "1.5, 1, Stuff, 8, -4.5" + '\n' +
                "2.5, 3, Stuff, -8, 24.5" + '\n' +
                "3.5, 5, Stuff, 18, 34.5" + '\n' +
                "4.5, 7, Stuff, -18, 44.5" + '\n' +
                "5.5, 215, Stuff, 81, 564.5" + '\n';
        Reader reader = new StringReader(fakeCSV);
        return new CSVParser(reader, CSVFormat.DEFAULT.withFirstRecordAsHeader().withIgnoreSurrou
ndingSpaces());
    }

    @Test
    public void parserCheck() throws IOException {
        Assertions.assertDoesNotThrow(() -> {
            CSVParser parser = getParser();
            List<CSVRecord> records = parser.getRecords();
            Assertions.assertEquals(5, records.size());
            List<String> headers = parser.getHeaderNames();
            Assertions.assertEquals(5, headers.size());
            Assertions.assertEquals("1.5", records.get(0).get(value1));
            Assertions.assertEquals("564.5", records.get(4).get(value5));
        });
    }
    @Test
    public void noPropertyNameThrows() {
        Assertions.assertThrows(NullPointerException.class, () -> {
            DoubleConstraint constraint = builder()
                    .lowerBound(6)
                    .upperBound(13).build();
            DoubleFilter df = new DoubleFilter(constraint);
        });
    }
    @Test
    public void singleMatchTest() {
```

```
        Assertions.assertDoesNotThrow(() -> {
            DoubleConstraint constraint = builder()
                    .propertyName(value1)
                    .lowerBound(2.73)
                    .upperBound(4.01).build();
            Filter filter = FilterFactory.from(constraint);
            List<CSVRecord> records = getParser().getRecords();
            Assertions.assertFalse(filter.matches(records.get(0)));
            Assertions.assertFalse(filter.matches(records.get(1)));
            Assertions.assertTrue(filter.matches(records.get(2)));
            Assertions.assertFalse(filter.matches(records.get(3)));
            Assertions.assertFalse(filter.matches(records.get(4)));
        });
        Assertions.assertDoesNotThrow(() -> {
            DoubleConstraint constraint = builder()
                    .propertyName(value5)
                    .lowerBound(-20)
                    .upperBound(0).build();
            Filter filter = FilterFactory.from(constraint);
            List<CSVRecord> records = getParser().getRecords();
            Assertions.assertTrue(filter.matches(records.get(0)));
            Assertions.assertFalse(filter.matches(records.get(1)));
            Assertions.assertFalse(filter.matches(records.get(2)));
            Assertions.assertFalse(filter.matches(records.get(3)));
            Assertions.assertFalse(filter.matches(records.get(4)));
        });
        Assertions.assertDoesNotThrow(() -> {
            DoubleConstraint constraint = builder()
                    .propertyName(value5)
                    .lowerBound(20)
                    .upperBound(-20).build();
            Filter filter = FilterFactory.from(constraint);
            List<CSVRecord> records = getParser().getRecords();
            Assertions.assertTrue(filter.matches(records.get(0)));
            Assertions.assertFalse(filter.matches(records.get(1)));
            Assertions.assertFalse(filter.matches(records.get(2)));
            Assertions.assertFalse(filter.matches(records.get(3)));
            Assertions.assertFalse(filter.matches(records.get(4)));
        });
    }
    @Test
    public void factoryWorks() {
        Assertions.assertDoesNotThrow(() -> {
            String json =
                    String.format("{ \"class\":\"mil.devcom_sc.ansur.api.constraints.DoubleConstr
aint\",\"propertyName\":\"%s\", \"lowerBound\":3.9, \"upperBound\":4.9}", value1);
            DoubleConstraint constraint = new JsonLoader().load(json, mil.devcom_sc.ansur.api.con
straints.DoubleConstraint.class, Path.of("Bob"));
            Filter filter = FilterFactory.from(constraint);
            List<CSVRecord> records = getParser().getRecords();
            Assertions.assertFalse(filter.matches(records.get(0)));
            Assertions.assertFalse(filter.matches(records.get(1)));
            Assertions.assertFalse(filter.matches(records.get(2)));
            Assertions.assertTrue(filter.matches(records.get(3)));
            Assertions.assertFalse(filter.matches(records.get(4)));
        });
    }
    @Test
    public void noPropertyNameThrow() {
        Assertions.assertThrows(SSTAFException.class, () -> {
            String json = "{ \"class\":\"mil.devcom_sc.ansur.api.constraints.DoubleFilter\", \"lo
werBound\":3.9, \"upperBound\":4.9}";
            mil.devcom_sc.ansur.api.constraints.DoubleConstraint constraint = new JsonLoader().lo
ad(json, mil.devcom_sc.ansur.api.constraints.DoubleConstraint.class, Path.of("Bob"));
        });
    }
    @Test
    public void builderWorks() {
```

```
        Assertions.assertDoesNotThrow(() -> {
            DoubleConstraint constraint = builder()
                    .propertyName(value1)
                    .lowerBound(3.9)
                    .upperBound(500)
                    .build();
            Filter filter = FilterFactory.from(constraint);
            List<CSVRecord> records = getParser().getRecords();
            Assertions.assertFalse(filter.matches(records.get(0)));
            Assertions.assertFalse(filter.matches(records.get(1)));
            Assertions.assertFalse(filter.matches(records.get(2)));
            Assertions.assertTrue(filter.matches(records.get(3)));
            Assertions.assertTrue(filter.matches(records.get(4)));
        });
    }

    @Test
    public void equalsWorks() {
        Assertions.assertDoesNotThrow(() -> {
            DoubleConstraint constraint = builder()
                    .propertyName(value1)
                    .equals(4.5)
                    .build();
            Filter filter = FilterFactory.from(constraint);
            List<CSVRecord> records = getParser().getRecords();
            Assertions.assertFalse(filter.matches(records.get(0)));
            Assertions.assertFalse(filter.matches(records.get(1)));
            Assertions.assertFalse(filter.matches(records.get(2)));
            Assertions.assertTrue(filter.matches(records.get(3)));
            Assertions.assertFalse(filter.matches(records.get(4)));
        });
    }
    @Test
    public void equalsWorks2() {
        Assertions.assertDoesNotThrow(() -> {
            DoubleConstraint constraint = builder()
                    .propertyName(value1)
                    .lowerBound(4.5)
                    .upperBound(4.5)
                    .build();
            Filter filter = FilterFactory.from(constraint);
            List<CSVRecord> records = getParser().getRecords();
            Assertions.assertFalse(filter.matches(records.get(0)));
            Assertions.assertFalse(filter.matches(records.get(1)));
            Assertions.assertFalse(filter.matches(records.get(2)));
            Assertions.assertTrue(filter.matches(records.get(3)));
            Assertions.assertFalse(filter.matches(records.get(4)));
        });
    }
    @Test
    public void badPropertyNameThrows() {
        Assertions.assertThrows(SSTAFException.class, () -> {
            DoubleConstraint constraint =
                    builder().propertyName("NotAValidProperty")
                            .lowerBound(1.0)
                            .upperBound(3.14)
                            .build();
            Filter filter = FilterFactory.from(constraint);
            List<CSVRecord> records = getParser().getRecords();
            Assertions.assertFalse(filter.matches(records.get(0)));
            Assertions.assertFalse(filter.matches(records.get(1)));
            Assertions.assertFalse(filter.matches(records.get(2)));
            Assertions.assertFalse(filter.matches(records.get(3)));
            Assertions.assertFalse(filter.matches(records.get(4)));
        });
    }
}
```

A.34 SSTAF/src/features/ansurHandler/mil.devcom_sc.ansur.handler/src/test/java/mil/devcom_sc/ansur/handler/filters/IntegerFilterTest.java

```
package mil.devcom_sc.ansur.handler.filters;
import mil.devcom_sc.ansur.api.constraints.IntegerConstraint;
import mil.devcom_sc.ansur.messages.ValueKey;
import mil.sstaf.core.json.JsonLoader;
import mil.sstaf.core.util.SSTAFException;
import org.apache.commons.csv.CSVFormat;
import org.apache.commons.csv.CSVParser;
import org.apache.commons.csv.CSVRecord;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;
import java.io.IOException;
import java.io.Reader;
import java.io.StringReader;
import java.nio.file.Path;
import java.util.List;
public class IntegerFilterTest {
    String value1 = ValueKey.EAR_LENGTH.getHeaderLabel();
    String value2 = ValueKey.FOOT_LENGTH.getHeaderLabel();
    String value3 = ValueKey.SUBJECTS_BIRTH_LOCATION.getHeaderLabel();
    String value4 = ValueKey.WEIGHT_LBS.getHeaderLabel();
    String value5 = ValueKey.WAIST_DEPTH.getHeaderLabel();
    CSVParser getParser() throws IOException {
        String fakeCSV
                = String.format("%s,%s,%s,%s,%s\n",
                value1, value2, value3, value4, value5) +
                "1.5, 1, Stuff, 8, -4.5" + '\n' +
                "2.5, 3, Stuff, -8, 24.5" + '\n' +
                "3.5, 5, Stuff, 18, 34.5" + '\n' +
                "4.5, 7, Stuff, -18, 44.5" + '\n' +
                "5.5, 215, Stuff, 81, 564.5" + '\n';
        Reader reader = new StringReader(fakeCSV);
        return new CSVParser(reader, CSVFormat.DEFAULT.withFirstRecordAsHeader().withIgnoreSurrou
ndingSpaces());
    }

    @Test
    public void parserCheck() throws IOException {
        Assertions.assertDoesNotThrow(() -> {
            CSVParser parser = getParser();
            List<CSVRecord> records = parser.getRecords();
            Assertions.assertEquals(5, records.size());
            List<String> headers = parser.getHeaderNames();
            Assertions.assertEquals(5, headers.size());
            Assertions.assertEquals("1.5", records.get(0).get(value1));
            Assertions.assertEquals("564.5", records.get(4).get(value5));
        });
    }
    @Test
    public void noPropertyNameThrows() {
        Assertions.assertThrows(NullPointerException.class, () -> {
            IntegerConstraint constraint = IntegerConstraint.builder()
                    .lowerBound(6)
                    .upperBound(13).build();
        });
    }
    @Test
    public void singleMatchTest() {
        Assertions.assertDoesNotThrow(() -> {
            IntegerConstraint constraint = IntegerConstraint.builder()
                    .propertyName(value2)
```

```
                    .lowerBound(6)
                    .upperBound(13).build();
            Filter filter = FilterFactory.from(constraint);
            List<CSVRecord> records = getParser().getRecords();
            Assertions.assertFalse(filter.matches(records.get(0)));
            Assertions.assertFalse(filter.matches(records.get(1)));
            Assertions.assertFalse(filter.matches(records.get(2)));
            Assertions.assertTrue(filter.matches(records.get(3)));
            Assertions.assertFalse(filter.matches(records.get(4)));
        });
        Assertions.assertDoesNotThrow(() -> {
            IntegerConstraint constraint = IntegerConstraint.builder()
                    .propertyName(value4)
                    .lowerBound(-20)
                    .upperBound(0).build();
            Filter filter = FilterFactory.from(constraint);
            List<CSVRecord> records = getParser().getRecords();
            Assertions.assertFalse(filter.matches(records.get(0)));
            Assertions.assertTrue(filter.matches(records.get(1)));
            Assertions.assertFalse(filter.matches(records.get(2)));
            Assertions.assertTrue(filter.matches(records.get(3)));
            Assertions.assertFalse(filter.matches(records.get(4)));
        });
        Assertions.assertDoesNotThrow(() -> {
            IntegerConstraint constraint = IntegerConstraint.builder()
                    .propertyName(value4)
                    .lowerBound(20)
                    .upperBound(-20).build();
            Filter filter = FilterFactory.from(constraint);
            List<CSVRecord> records = getParser().getRecords();
            Assertions.assertTrue(filter.matches(records.get(0)));
            Assertions.assertTrue(filter.matches(records.get(1)));
            Assertions.assertTrue(filter.matches(records.get(2)));
            Assertions.assertTrue(filter.matches(records.get(3)));
            Assertions.assertFalse(filter.matches(records.get(4)));
        });
    }
    @Test
    public void factoryWorks() {
        Assertions.assertDoesNotThrow(() -> {
            String json = String.format("{ \"class\":\"mil.devcom_sc.ansur.api.constraints.Intege
rConstraint\",\"propertyName\":\"%s\", \"lowerBound\":6, \"upperBound\":13}", value2);
            IntegerConstraint constraint = new JsonLoader().load(json, IntegerConstraint.class, P
ath.of("bob"));
            Filter filter = FilterFactory.from(constraint);
            List<CSVRecord> records = getParser().getRecords();
            Assertions.assertFalse(filter.matches(records.get(0)));
            Assertions.assertFalse(filter.matches(records.get(1)));
            Assertions.assertFalse(filter.matches(records.get(2)));
            Assertions.assertTrue(filter.matches(records.get(3)));
            Assertions.assertFalse(filter.matches(records.get(4)));
        });
    }
    @Test
    public void missingPropertyThrows() {
        Assertions.assertThrows(SSTAFException.class, () -> {
            String json = "{ \"class\":\"mil.devcom_sc.ansur.api.constraints.IntegerConstraint\",
\"lowerBound\":6, \"upperBound\":13}";
            IntegerConstraint constraint = new JsonLoader().load(json, IntegerConstraint.class, P
ath.of("bob"));
        });
    }
    @Test
    public void equalsWorks() {
        Assertions.assertDoesNotThrow(() -> {
            IntegerConstraint constraint = IntegerConstraint.builder()
                    .propertyName(value2)
                    .equals(7)
```

```
                    .build();
            Filter filter = FilterFactory.from(constraint);
            List<CSVRecord> records = getParser().getRecords();
            Assertions.assertFalse(filter.matches(records.get(0)));
            Assertions.assertFalse(filter.matches(records.get(1)));
            Assertions.assertFalse(filter.matches(records.get(2)));
            Assertions.assertTrue(filter.matches(records.get(3)));
            Assertions.assertFalse(filter.matches(records.get(4)));
        });
    }
    @Test
    public void badPropertyNameThrows() {
        Assertions.assertThrows(SSTAFException.class, () -> {
            IntegerConstraint constraint = IntegerConstraint.builder()
                    .propertyName("ThisIsBad")
                    .equals(7)
                    .build();
            Filter filter = FilterFactory.from(constraint);
            List<CSVRecord> records = getParser().getRecords();
            Assertions.assertFalse(filter.matches(records.get(0)));
            Assertions.assertFalse(filter.matches(records.get(1)));
            Assertions.assertFalse(filter.matches(records.get(2)));
            Assertions.assertFalse(filter.matches(records.get(3)));
            Assertions.assertFalse(filter.matches(records.get(4)));
        });
    }
}
```

A.35 SSTAF/src/features/ansurHandler/mil.devcom_sc.ansur.handler/src/test/java/mil/devcom_sc/ansur/handler/filters/StringFilterTest.java

```
package mil.devcom_sc.ansur.handler.filters;
import mil.devcom_sc.ansur.api.constraints.StringConstraint;
import mil.devcom_sc.ansur.messages.ValueKey;
import mil.sstaf.core.json.JsonLoader;
import mil.sstaf.core.util.SSTAFException;
import org.apache.commons.csv.CSVFormat;
import org.apache.commons.csv.CSVParser;
import org.apache.commons.csv.CSVRecord;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;
import java.io.IOException;
import java.io.Reader;
import java.io.StringReader;
import java.nio.file.Path;
import java.util.List;
public class StringFilterTest {
    String value1 = ValueKey.EAR_LENGTH.getHeaderLabel();
    String value2 = ValueKey.FOOT_LENGTH.getHeaderLabel();
    String value3 = ValueKey.SUBJECTS_BIRTH_LOCATION.getHeaderLabel();
    String value4 = ValueKey.WEIGHT_LBS.getHeaderLabel();
    String value5 = ValueKey.WAIST_DEPTH.getHeaderLabel();
    CSVParser getParser() throws IOException {
        String fakeCSV
                = String.format("%s,%s,%s,%s,%s\n",
                value1, value2, value3, value4, value5) +
                "1.5, 1, Stuff, 8, -4.5" + '\n' +
                "2.5, 3, also_banana, -8, 24.5" + '\n' +
                "3.5, 5, banana, 18, 34.5" + '\n' +
                "4.5, 7, Stuff, -18, 44.5" + '\n' +
                "5.5, 215, Wilma, 81, 564.5" + '\n';
        Reader reader = new StringReader(fakeCSV);
        return new CSVParser(reader, CSVFormat.DEFAULT.withFirstRecordAsHeader().withIgnoreSurrou
ndingSpaces());
    }

    @Test
    public void parserCheck() throws IOException {
        Assertions.assertDoesNotThrow(() -> {
            CSVParser parser = getParser();
            List<CSVRecord> records = parser.getRecords();
            Assertions.assertEquals(5, records.size());
            List<String> headers = parser.getHeaderNames();
            Assertions.assertEquals(5, headers.size());
            Assertions.assertEquals("1.5", records.get(0).get(value1));
            Assertions.assertEquals("banana", records.get(2).get(value3));
            Assertions.assertEquals("564.5", records.get(4).get(value5));
        });
    }
    @Test
    public void noPropertyNameThrows() {
        Assertions.assertThrows(NullPointerException.class, () -> {
            StringConstraint constraint = StringConstraint.builder()
                    .matches("banana").build();
        });
    }
    @Test
    public void singleMatchTest() {
        Assertions.assertDoesNotThrow(() -> {
            StringConstraint constraint = StringConstraint.builder()
                    .propertyName(value3)
```

```
                    .matches("Wilma").build();
            Filter filter = FilterFactory.from(constraint);
            List<CSVRecord> records = getParser().getRecords();
            Assertions.assertFalse(filter.matches(records.get(0)));
            Assertions.assertFalse(filter.matches(records.get(1)));
            Assertions.assertFalse(filter.matches(records.get(2)));
            Assertions.assertFalse(filter.matches(records.get(3)));
            Assertions.assertTrue(filter.matches(records.get(4)));
        });
    }
    @Test
    public void factoryWorks() {
        Assertions.assertDoesNotThrow(() -> {
            String json = String.format("{ \"class\":\"mil.devcom_sc.ansur.api.constraints.String
Constraint\", \"propertyName\":\"%s\", \"matches\":\".*anana.*\"}", value3);
            StringConstraint constraint =
                    new JsonLoader().load(json, StringConstraint.class, Path.of("Fake.json"));
            Filter filter = FilterFactory.from(constraint);
            List<CSVRecord> records = getParser().getRecords();
            Assertions.assertFalse(filter.matches(records.get(0)));
            Assertions.assertTrue(filter.matches(records.get(1)));
            Assertions.assertTrue(filter.matches(records.get(2)));
            Assertions.assertFalse(filter.matches(records.get(3)));
            Assertions.assertFalse(filter.matches(records.get(4)));
        });
    }
    @Test
    public void badPropertyNameThrows() {
        Assertions.assertThrows(SSTAFException.class, () -> {
            String json = "{ \"class\":\"mil.devcom_sc.ansur.api.constraints.StringConstraint\",
\"propertyName\":\"Widget\", \"matches\":\".*anana.*\"}";
            StringConstraint constraint =
                    new JsonLoader().load(json, StringConstraint.class, Path.of("Fake.json"));
            Filter filter = FilterFactory.from(constraint);
            List<CSVRecord> records = getParser().getRecords();
            Assertions.assertFalse(filter.matches(records.get(0)));
            Assertions.assertFalse(filter.matches(records.get(1)));
            Assertions.assertFalse(filter.matches(records.get(2)));
            Assertions.assertFalse(filter.matches(records.get(3)));
            Assertions.assertFalse(filter.matches(records.get(4)));
        });
    }
}
```

A.36 SSTAF/src/features/ansurHandler/mil.devcom_sc.ansur.handler/src/test/resources/FemalesUnder65Kg.json

```
{
  "class" : "mil.devcom_sc.ansur.api.ANSURConfiguration",
  "constraints": [
    {
      "class" : "mil.devcom_sc.ansur.api.constraints.StringConstraint",
      "propertyName": "Gender",
      "matches": "Female"
    },
    {
      "class" : "mil.devcom_sc.ansur.api.constraints.IntegerConstraint",
      "propertyName": "weightkg",
      "upperBound" : 650
    }
  ]
}
```

A.37 SSTAF/src/features/ansurHandler/mil.devcom_sc.ansur.handler/src/test/resources/log4j2-test.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="INFO">
    <Appenders>
        <Console name="Console" target="SYSTEM_OUT">
            <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"/>
        </Console>
    </Appenders>
    <Loggers>
        <Root level="trace">
            <AppenderRef ref="Console"/>
        </Root>
    </Loggers>
</Configuration>
```

A.38 SSTAF/src/features/ansurHandler/mil.devcom_sc.ansur.handler/src/test/resources/multiple_constraints.json

```
{
  "class" : "mil.devcom_sc.ansur.api.ANSURConfiguration",
  "constraints": [
    {
      "class" : "mil.devcom_sc.ansur.api.constraints.StringConstraint",
      "propertyName": "Gender",
      "matches": "Male"
    },
    {
      "class" : "mil.devcom_sc.ansur.api.constraints.IntegerConstraint",
      "propertyName": "balloffootlength",
      "equals": 198
    },
    {
      "class" : "mil.devcom_sc.ansur.api.constraints.IntegerConstraint",
      "propertyName": "footlength",
      "equals": 270
    },
    {
      "class" : "mil.devcom_sc.ansur.api.constraints.IntegerConstraint",
      "propertyName": "lowerthighcircumference",
      "lowerBound": 390,
      "upperBound": 395
    },
    {
      "class" : "mil.devcom_sc.ansur.api.constraints.StringConstraint",
      "propertyName": "SubjectsBirthLocation",
      "matches": ".*uck.*"
    }
  ]
}
```

A.39 SSTAF/src/features/ansurHandler/mil.devcom_sc.ansur.handler/src/test/resources/subjectID.json

```
{
  "class" : "mil.devcom_sc.ansur.api.constraints.IntegerConstraint",
  "propertyName": "subjectID",
  "equals" : 10173
}
```

A.40 SSTAF/src/features/ansurHandler/mil.devcom_sc.ansur.handler/src/test/resources/subjectID_as_constraint.json

```
{
  "class": "mil.devcom_sc.ansur.api.constraints.IntegerConstraint",
  "propertyName": "subjectID",
  "lowerBound": 10173,
  "upperBound": 10173
}
```

A.41 SSTAF/src/features/ansurHandler/mil.devcom_sc.ansur.messages/build.gradle

```
dependencies {
    implementation project(':framework:mil.sstaf.core')
    implementation 'org.projectlombok:lombok:1.18.20'

    testRuntimeOnly project(":features:support:mil.sstaf.blackboard.inmem")
}
```

A.42 SSTAF/src/features/ansurHandler/mil.devcom_sc.ansur.messages/src/main/java/mil/devcom_sc/ansur/messages/GetValueMessage.java

```
package mil.devcom_sc.ansur.messages;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import mil.sstaf.core.features.HandlerContent;
import java.util.Objects;
/**
 * Command object used to request a value from the {@code Entity}'s instance of
 * {@code ANSURIIAnthropometry}.
 */
@SuperBuilder
@Jacksonized
public class GetValueMessage extends HandlerContent {
    public final ValueKey key;
    private GetValueMessage(GetValueMessageBuilder<?, ?> builder) {
        super(builder);
        Objects.requireNonNull(builder.key, "key");
        key = builder.key;
    }
    /**
     * Factory for making the message
     *
     * @param key the {@code ValueKey} being requested
     * @return a new message object
     */
    public static GetValueMessage of(ValueKey key) {
        Objects.requireNonNull(key, "Key must not be null");
        return builder().key(key).build();
    }
}
```

A.43 SSTAF/src/features/ansurHandler/mil.devcom_sc.ansur.messages/src/main/java/mil/devcom_sc/ansur/messages/GetValueResponse.java

```
package mil.devcom_sc.ansur.messages;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.EqualsAndHashCode;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import mil.sstaf.core.features.HandlerContent;
import mil.sstaf.core.util.SSTAFException;
import java.util.Objects;
import java.util.Optional;
/**
 * The response from a {@code GetValueMessage}
 */
@SuperBuilder
@Jacksonized
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
@EqualsAndHashCode(callSuper = true)
public class GetValueResponse extends HandlerContent {
    private final ValueKey valueKey;
    private final String stringValue;
    private final double doubleValue;
    private final int intValue;
    /**
     * Constructor
     */
    private GetValueResponse(GetValueResponseBuilder<?,?> builder) {
        super(builder);
        Objects.requireNonNull(builder.valueKey);
        valueKey = builder.valueKey;
        if (builder.valueKey.getType().equals(String.class)) {
            Objects.requireNonNull(builder.stringValue);
            stringValue = builder.stringValue;
            doubleValue = Double.MIN_VALUE;
            intValue = Integer.MIN_VALUE;
        } else if (builder.valueKey.getType().equals(Double.class)) {
            doubleValue = builder.doubleValue;
            intValue = Integer.MIN_VALUE;
            stringValue = null;
        } else if (builder.valueKey.getType().equals(Integer.class)) {
            intValue = builder.intValue;
            doubleValue = Double.MIN_VALUE;
            stringValue = null;
        } else {
            throw new SSTAFException("Value key has unsupported type");
        }
    }
    /**
     * Factory method for making a GetValueResponse with a String.
     *
     * @param value the value to include
     * @return the response object.
     */
    public static GetValueResponse of(final ValueKey valueKey, final String value) {
        Objects.requireNonNull(valueKey);
        if (!valueKey.getType().equals(String.class)) {
            throw new SSTAFException("ValueKey type is not a String");
        }
        return builder().valueKey(valueKey).stringValue(value).build();
    }
    public static GetValueResponse of(final ValueKey valueKey, final double value) {
        Objects.requireNonNull(valueKey);
```

```
        if (!valueKey.getType().equals(Double.class)) {
            throw new SSTAFException("ValueKey type is not a double");
        }
        return builder().valueKey(valueKey).doubleValue(value).build();
    }
    public static GetValueResponse of(final ValueKey valueKey, final int value) {
        Objects.requireNonNull(valueKey);
        if (! valueKey.getType().equals(Integer.class)) {
            throw new SSTAFException("ValueKey type is not a double");
        }
        return builder().valueKey(valueKey).intValue(value).build();
    }
    /**
     * Returns the value wrapped in an {@code Optional}.
     *
     * @return an {@code Optional} containing the value or empty.
     */
    public Optional<Object> getValue() {
        if (valueKey.getType().equals(String.class)) {
           return Optional.of(stringValue);
        } else if (valueKey.getType().equals(Double.class)) {
            return Optional.of(doubleValue);
        } else if (valueKey.getType().equals(Integer.class)) {
           return Optional.of(intValue);
        } else {
            return Optional.empty();
        }
    }
    /**
     * Provides the value wrapped in an {@code Optional} if it is a {@code String}.
     *
     * @return an {@code Optional} containing the value or empty.
     */
    public Optional<String> getStringValue() {
        return Optional.ofNullable(stringValue);
    }
    /**
     * Provides the value wrapped in an {@code Optional} if it is a {@code Double}.
     *
     * @return an {@code Optional} containing the value or empty.
     */
    public Optional<Double> getDoubleValue() {
        if (valueKey.getType().equals(Double.class)) {
            return Optional.of(doubleValue);
        } else {
            return Optional.empty();
        }
    }
    /**
     * Provides the value wrapped in an {@code Optional} if it is a {@code Integer}.
     *
     * @return an {@code Optional} containing the value or empty.
     */
    public Optional<Integer> getIntegerValue() {
        if (valueKey.getType().equals(Integer.class)) {
            return Optional.of(intValue);
        } else {
            return Optional.empty();
        }
    }
}
```

A.44 SSTAF/src/features/ansurHandler/mil.devcom_sc.ansur.messages/src/main/java/mil/devcom_sc/ansur/messages/Handedness.java

```
package mil.devcom_sc.ansur.messages;
public enum Handedness {
    LEFT,
    RIGHT
}
```

A.45 SSTAF/src/features/ansurHandler/mil.devcom_sc.ansur.messages/src/main/java/mil/devcom_sc/ansur/messages/Sex.java

```
package mil.devcom_sc.ansur.messages;
/**
 * Enum for specifying the Sex of an {@code Entity}.
 */
public enum Sex {
    FEMALE,
    MALE
}
```

A.46 SSTAF/src/features/ansurHandler/mil.devcom_sc.ansur.messages/src/main/java/mil/devcom_sc/ansur/messages/ValueKey.java

```
package mil.devcom_sc.ansur.messages;
import lombok.extern.jackson.Jacksonized;
import mil.sstaf.core.util.SSTAFException;
/**
 * Enumeration of all of the fields in the ANSUR II database.
 * <p>
 * Note that the names of the keys do not match the headers in the CSV files. Most of the headers
are all lower-case
 * and difficult to read, so each key includes a separate headerLabel value.
 */
public enum ValueKey {
    SUBJECT_ID("subjectid",
            Integer.class,
            "A unique number for each participant measured in the anthropometric survey, ranging
from 10027 to 920103, not inclusive"),
    //
    // Note the inconsistent use of Subject/Subjects (implied "'s"). This is necessary
    // to be consistent with the data files.
    //
    SUBJECTS_BIRTH_LOCATION("SubjectsBirthLocation",
            String.class,
            "Subject's Birth Location; a U.S. state or foreign country"),
    SUBJECT_NUMERIC_RACE("SubjectNumericRace",
            Integer.class,
            "Subject Numeric Race; a single or multi-digit code indicating a "
                    + "subject’s self-reported race or races (verified through interview). Where
1 = White, "
                    + "2 = Black, 3 = Hispanic, 4 = Asian, 5 = Native American, 6 = Pacific Islan
der, 8 = Other"),
    ETHNICITY("Ethnicity",
            String.class,
            "self-reported ethnicity (verified through interview); e.g. 'Mexican', 'Vietnamese'")
,
    DOD_RACE("DODRace",
            Integer.class,
            "Department of Defense Race; a single digit indicating a subject’s self-reported pref
erred single race where "
                    + "selecting multiple races is not an option. This variable is intended to be
comparable to the "
                    + "Defense Manpower Data Center demographic data. Where 1 = White, 2 = Black,
3 = Hispanic, 4 = Asian,"
                    + "5 = Native American, 6 = Pacific Islander, 8 = Other"),
    GENDER("Gender",
            String.class,
            "'Male' or 'Female'"),
    AGE("Age",
            Integer.class,
            "Participant’s age in years"),
    HEIGHT_IN("Heightin",
            Integer.class,
            "Height in Inches; self-reported, comparable to measured 'stature'"),
    WEIGHT_LBS("Weightlbs",
            Integer.class,
            "Weight in Pounds; self-reported, comparable to measured 'weightkg'"),
    WRITING_PREFERENCE("WritingPreference",
            String.class,
            "Writing Preference; 'Right hand', 'Left hand', or " +
                    "'Either hand (No preference)'"),
    DATE("Date",
            String.class,
```

```
            "Date the participant was measured, ranging from '04-Oct-10' to '05-Apr-12'"),
    INSTALLATION("Installation",
            String.class,
            " U.S. Army installation where the measurement occurred, e.g. 'Fort Hood', 'Camp Shel
by'"),
    COMPONENT("Component",
            String.class,
            "'Army National Guard','Army Reserve', or 'Regular Army'"),
    BRANCH("Branch",
            String.class,
            "'Combat Arms', 'Combat Support', or 'Combat Service Support'"),
    PRIMARY_MOS("PrimaryMOS",
            String.class,
            "Primary Military Occupational Specialty"),
    ABDOMINAL_EXTENSION_DEPTH_SITTING("abdominalextensiondepthsitting",
            Integer.class,
            "Abdominal Extension Depth, Sitting"),
    ACROMIAL_HEIGHT("acromialheight",
            Integer.class,
            "Acromial Height"),
    ACROMION_RADIALE_LENGTH("acromionradialelength",
            Integer.class,
            "Acromion-Radiale Length"),
    ANKLE_CIRCUMFERENCE("anklecircumference",
            Integer.class,
            "Ankle Circumference"),
    AXILLA_HEIGHT("axillaheight",
            Integer.class,
            "Axilla Height"),
    BALL_OF_FOOT_CIRCUMFERENCE("balloffootcircumference",
            Integer.class,
            "Ball of Foot Circumference"),
    BALL_OF_FOOT_LENGTH("balloffootlength",
            Integer.class,
            "Ball of Foot Length"),
    BIACROMIAL_BREADTH("biacromialbreadth",
            Integer.class,
            "Biacromial Breadth"),
    BICEPS_CIRCUMFERENCE_FLEXED("bicepscircumferenceflexed",
            Integer.class,
            "Biceps Circumference, Flexed"),
    BICRISTAL_BREADTH("bicristalbreadth",
            Integer.class,
            "Bicristal Breadth"),
    BIDELTOID_BREADTH("bideltoidbreadth",
            Integer.class,
            "Bideltoid Breadth"),
    BIMALLEOLAR_BREADTH("bimalleolarbreadth",
            Integer.class,
            "Bimalleolar Breadth"),
    BITRAGION_CHIN_ARC("bitragionchinarc",
            Integer.class,
            "Bitragion Chin Arc"),
    BITRAGION_SUBMANDIBULAR_ARC("bitragionsubmandibulararc",
            Integer.class,
            "Bitragion Submandibular Arc"),
    BIZYGOMATIC_BREADTH("bizygomaticbreadth",
            Integer.class,
            "Bizygomatic Breadth"),
    BUTTOCK_CIRCUMFERENCE("buttockcircumference",
            Integer.class,
            "Buttock Circumference"),
    BUTTOCK_DEPTH("buttockdepth",
            Integer.class,
            "Buttock Depth"),
    BUTTOCK_HEIGHT("buttockheight",
            Integer.class,
            "Buttock Height"),
```

```
    BUTTOCK_KNEE_LENGTH("buttockkneelength",
            Integer.class,
            "Buttock-Knee Length"),
    BUTTOCK_POPLITEAL_LENGTH("buttockpopliteallength",
            Integer.class,
            "Buttock-Popliteal Length"),
    CALF_CIRCUMFERENCE("calfcircumference",
            Integer.class,
            "Calf Circumference"),
    CERVICALE_HEIGHT("cervicaleheight",
            Integer.class,
            "Cervical Height"),
    CHEST_BREADTH("chestbreadth",
            Integer.class,
            "Chest Breadth"),
    CHEST_CIRCUMFERENCE("chestcircumference",
            Integer.class,
            "Chest Circumference"),
    CHEST_DEPTH("chestdepth",
            Integer.class,
            "Chest Depth"),
    CHEST_HEIGHT("chestheight",
            Integer.class,
            "Chest Height"),
    CROTCH_HEIGHT("crotchheight",
            Integer.class,
            "Crotch Height"),
    CROTCH_LENGTH_OMPHALION("crotchlengthomphalion",
            Integer.class,
            "Crotch Length (Omphalion)"),
    CROTCH_LENGTH_POSTERIOR_OMPHALION("crotchlengthposteriomphalion",
            Integer.class,
            "Crotch Length, Posterior (Omphalion)"),
    EAR_BREADTH("earbreadth",
            Integer.class,
            "Ear Breadth"),
    EAR_LENGTH("earlength",
            Integer.class,
            "Ear Length"),
    EAR_PROTRUSION("earprotrusion",
            Integer.class,
            "Ear Protrusion"),
    ELBOW_REST_HEIGHT("elbowrestheight",
            Integer.class,
            "Elbow Rest Height"),
    EYE_HEIGHT_SITTING("eyeheightsitting",
            Integer.class,
            "Eye Height, Sitting"),
    FOOT_BREADTH_HORIZONTAL("footbreadthhorizontal",
            Integer.class,
            "Foot Breadth, Horizontal"),
    FOOT_LENGTH("footlength",
            Integer.class,
            "Foot Length"),
    FOREARM_CENTER_OF_GRIP_LENGTH("forearmcenterofgriplength",
            Integer.class,
            "Forearm-Center of Grip Length"),
    FOREARM_CIRCUMFERENCE_FLEXED("forearmcircumferenceflexed",
            Integer.class,
            "Forearm Circumference, Flexed"),
    FOREARM_FOREARM_BREADTH("forearmforearmbreadth",
            Integer.class,
            "Forearm-Forearm Breadth"),
    FOREARM_HAND_LENGTH("forearmhandlength",
            Integer.class,
            "Forearm -Hand Length"),
    FUNCTIONAL_LEG_LENGTH("functionalleglength",
            Integer.class,
```

```
            "Functional Leg Length"),
    HAND_BREADTH("handbreadth",
            Integer.class,
            "Hand Breadth"),
    HAND_CIRCUMFERENCE("handcircumference",
            Integer.class,
            "Hand Circumference"),
    HAND_LENGTH("handlength",
            Integer.class,
            "Hand Length"),
    HEAD_BREADTH("headbreadth",
            Integer.class,
            "Head Breadth"),
    HEAD_CIRCUMFERENCE("headcircumference",
            Integer.class,
            "Head Circumference"),
    HEAD_LENGTH("headlength",
            Integer.class,
            "Head Length"),
    HEEL_ANKLE_CIRCUMFERENCE("heelanklecircumference",
            Integer.class,
            "Heel-Ankle Circumference"),
    HEEL_BREADTH("heelbreadth",
            Integer.class,
            "Heel Breadth"),
    HIP_BREADTH("hipbreadth",
            Integer.class,
            "Hip Breadth"),
    HIP_BREADTH_SITTING("hipbreadthsitting",
            Integer.class,
            "Hip Breadth, Sitting"),
    ILIOCRISTALE_HEIGHT("iliocristaleheight",
            Integer.class,
            "Iliocristale Height"),
    INTERPUPILLARY_BREADTH("interpupillarybreadth",
            Integer.class,
            "Interpupillary Breadth"),
    INTERSCYE_I("interscyei",
            Integer.class,
            "Interscye I"),
    INTERSCYE_II("interscyeii",
            Integer.class,
            "Interscye II"),
    KNEE_HEIGHT_MIDPATELLA("kneeheightmidpatella",
            Integer.class,
            "Knee Height, Midpatella"),
    KNEE_HEIGHT_SITTING("kneeheightsitting",
            Integer.class,
            "Knee Height, Sitting"),
    LATERAL_FEMORAL_EPICONDYLE_HEIGHT("lateralfemoralepicondyleheight",
            Integer.class,
            "Lateral Femoral Epicondyle Height"),
    LATERAL_MALLEOLUS_HEIGHT("lateralmalleolusheight",
            Integer.class,
            "Lateral Malleolus Height"),
    LOWER_THIGH_CIRCUMFERENCE("lowerthighcircumference",
            Integer.class,
            "Lower Thigh Circumference"),
    MENTON_SELLION_LENGTH("mentonsellionlength",
            Integer.class,
            "Menton-Sellion Length"),
    NECK_CIRCUMFERENCE("neckcircumference",
            Integer.class,
            "Neck Circumference"),
    NECK_CIRCUMFERENCE_BASE("neckcircumferencebase",
            Integer.class,
            "Neck Circumference, Base"),
    OVERHEAD_FINGERTIP_REACH_SITTING("overheadfingertipreachsitting",
```

```
            Integer.class,
            "Overhead Fingertip Reach, Sitting"),
    PALM_LENGTH("palmlength",
            Integer.class,
            "Palm Length"),
    POPLITEAL_HEIGHT("poplitealheight",
            Integer.class,
            "Popliteal Height"),
    RADIALE_STYLION_LENGTH("radialestylionlength",
            Integer.class,
            "Radiale-Stylion Length"),
    SHOULDER_CIRCUMFERENCE("shouldercircumference",
            Integer.class,
            "Shoulder Circumference"),
    SHOULDER_ELBOW_LENGTH("shoulderelbowlength",
            Integer.class,
            "Shoulder-Elbow Length"),
    SHOULDER_LENGTH("shoulderlength",
            Integer.class,
            "Shoulder Length"),
    SITTING_HEIGHT("sittingheight",
            Integer.class,
            "Sitting Height"),
    SLEEVE_LENGTH_SPINE_WRIST("sleevelengthspinewrist",
            Integer.class,
            "Sleeve Length: Spine-Wrist"),
    SLEEVE_OUTSEAM("sleeveoutseam",
            Integer.class,
            "Sleeve Outseam"),
    SPAN("span",
            Integer.class,
            "Span"),
    STATURE("stature",
            Integer.class,
            "Stature"),
    SUPRASTERNALE_HEIGHT("suprasternaleheight",
            Integer.class,
            "Suprasternale Height"),
    TENTH_RIB_HEIGHT("tenthribheight",
            Integer.class,
            "Tenth Rib Height"),
    THIGH_CIRCUMFERENCE("thighcircumference",
            Integer.class,
            "Thigh Circumference"),
    THIGH_CLEARANCE("thighclearance",
            Integer.class,
            "Thigh Clearance"),
    THUMBTIP_REACH("thumbtipreach",
            Integer.class,
            "Thumbtip Reach"),
    TIBIAL_HEIGHT("tibialheight",
            Integer.class,
            "Tibiale Height"),
    TRAGION_TOP_O_FHEAD("tragiontopofhead",
            Integer.class,
            "Tragion-Top of Head"),
    TROCHANTERION_HEIGHT("trochanterionheight",
            Integer.class,
            "Trochanterion Height"),
    VERTICAL_TRUNK_CIRCUMFERENCE_USA("verticaltrunkcircumferenceusa",
            Integer.class,
            "Vertical Trunk Circumference (USA)"),
    WAISTBACK_LENGTH("waistbacklength",
            Integer.class,
            "Waist Back Length (Omphalion)"),
    WAIST_BREADTH("waistbreadth",
            Integer.class,
            "Waist Breadth"),
```

```
    WAIST_CIRCUMFERENCE("waistcircumference",
            Integer.class,
            "Waist Circumference (Omphalion)"),
    WAIST_DEPTH("waistdepth",
            Integer.class,
            "Waist Depth"),
    WAIST_FRONT_LENGTH_SITTING("waistfrontlengthsitting",
            Integer.class,
            "Waist Front Length, Sitting"),
    WAIST_HEIGHT_OMPHALION("waistheightomphalion",
            Integer.class,
            "Waist Height (Omphalion)"),
    WEIGHT_KG("weightkg",
            Integer.class,
            "Weight (in kg*10)"),
    WRIST_CIRCUMFERENCE("wristcircumference",
            Integer.class,
            "Wrist Circumference"),
    WRIST_HEIGHT("wristheight",
            Integer.class,
            "Wrist Height");
    private final String headerLabel;
    private final String description;
    private final Class<?> type;
    /**
     * Constructor
     *
     * @param headerLabel the name, also column header label
     * @param description the description
     */
    ValueKey(final String headerLabel, final Class<?> type, final String description) {
        this.headerLabel = headerLabel;
        this.type = type;
        this.description = description;
    }
    /**
     * Finds the ValueKey that matches the provided header name.
     *
     * @param name the name to match.
     * @return the matching ValueKey
     */
    public static ValueKey matchHeaderName(final String name) {
        for (ValueKey key : ValueKey.values()) {
            if (key.getHeaderLabel().equalsIgnoreCase(name)) {
                return key;
            }
        }
        throw new SSTAFException("Could not ANSUR ValueKey to match " + name);
    }
    /**
     * Provides the name of the measurement which is also the column label in the CSV file.
     *
     * @return the name
     */
    public String getHeaderLabel() {
        return headerLabel;
    }
    /**
     * Returns the type for this field.
     *
     * @return the type
     */
    public Class<?> getType() {
        return type;
    }
    /**
     * Provides the description of the measurement.
     *
```

```
     * @return the description
     */
    public String getDescription() {
        return description;
    }
}
```

A.47 SSTAF/src/features/ansurHandler/mil.devcom_sc.ansur.messages/src/main/java/module-info.java

```
module mil.devcom_sc.ansur.messages {
    exports mil.devcom_sc.ansur.messages;
    requires transitive mil.sstaf.core;
    requires lombok;
    requires com.fasterxml.jackson.databind;
    opens mil.devcom_sc.ansur.messages to com.fasterxml.jackson.databind;
}
```

A.48 SSTAF/src/features/ansurHandler/mil.devcom_sc.ansur.messages/src/test/java/mil/devcom_sc/ansur/messages/GetValueMessageTest.java

```
package mil.devcom_sc.ansur.messages;
import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Nested;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.MethodSource;
import java.util.Arrays;
import java.util.List;
import static org.junit.jupiter.api.Assertions.*;
public class GetValueMessageTest {
    static List<ValueKey> getMessages() {
        return Arrays.asList(ValueKey.values());
    }

    @Nested
    @DisplayName("Test the 'happy path'")
    class HappyTests {
        @ParameterizedTest(name = "{index} ==> Testing ''{0}''")
        @MethodSource("mil.devcom_sc.ansur.messages.GetValueMessageTest#getMessages")
        @DisplayName("Confirm that a GetValueMessage can be created, serialized and deserialized"
)
        void roundTrip(ValueKey valueKey) {
            ObjectMapper mapper = new ObjectMapper();
            GetValueMessage getValueMessage = GetValueMessage.builder().key(valueKey).build();
            assertNotNull(getValueMessage);
            assertEquals(valueKey, getValueMessage.key);
            assertDoesNotThrow(() -> {
                String json = mapper.writeValueAsString(getValueMessage);
                assertTrue(json.contains(valueKey.name()));
                JsonNode jsonNode = mapper.readTree(json);
                assertNotNull(jsonNode);
                GetValueMessage msg = mapper.treeToValue(jsonNode, GetValueMessage.class);
                assertNotNull(msg);
                assertNotNull(msg.key);
                assertEquals(valueKey, msg.key);
            });
        }
    }

    @Nested
    @DisplayName("Test failure modes")
    class FailureTests {
        @Test
        @DisplayName("Confirm that a null ValueKey throws")
        void test1() {
            NullPointerException npe = assertThrows(NullPointerException.class,
                    () -> {
                        GetValueMessage.builder().key(null).build();
                    });
            assertTrue(npe.getMessage().contains("key"));
        }
    }
}
```

A.49 SSTAF/src/features/equipment/mil.devcom_dac.equipment.api/build.gradle

```
dependencies {
    implementation project(':framework:mil.sstaf.core')
}
```

A.50 SSTAF/src/features/equipment/mil.devcom_dac.equipment.api/src/main/java/mil/devcom_dac/equipment/api/EquipmentConfiguration.java

```
package mil.devcom_dac.equipment.api;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.Builder;
import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import mil.sstaf.core.features.FeatureConfiguration;
import java.util.Objects;
@SuperBuilder
@Jacksonized
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
@EqualsAndHashCode(callSuper = true)
public class EquipmentConfiguration extends FeatureConfiguration {
    @Getter
    @Builder.Default
    private Kit kit = Kit.builder().build();
    @Getter
    private String currentGun;
}
```

A.51 SSTAF/src/features/equipment/mil.devcom_dac.equipment.api/src/main/java/mil/devcom_dac/equipment/api/EquipmentManagement.java

```
package mil.devcom_dac.equipment.api;
import mil.sstaf.core.features.Handler;
/**
 * Interface for managing equipment
 */
public interface EquipmentManagement extends SoldierKit, Handler {

}
```

A.52 SSTAF/src/features/equipment/mil.devcom_dac.equipment.api/src/main/java/mil/devcom_dac/equipment/api/Gun.java

```
package mil.devcom_dac.equipment.api;
import com.fasterxml.jackson.annotation.JsonIgnore;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.Builder;
import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
@SuperBuilder
@Jacksonized
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
@EqualsAndHashCode(callSuper = true)
public class Gun extends Item {
    @Getter
    private double emptyMass_kg;
    @Getter
    private String magazineType;
    @Getter
    @JsonIgnore
    @Builder.Default
    private Magazine magazine = null;
    private Gun(GunBuilder<?, ?> b) {
        super(b);
        this.emptyMass_kg = b.emptyMass_kg;
        this.magazineType = b.magazineType;
    }
    public double getMass_kg() {
        return emptyMass_kg + (magazine == null ? 0.0 : magazine.getMass_kg());
    }
    public void loadMagazine(final Magazine magazine) {
        this.magazine = magazine;
    }
    public void dropMagazine() {
        this.magazine = null;
    }
    public boolean canShoot() {
        return magazine != null && !(magazine.isEmpty());
    }
    public int shoot(int numToShoot) {
        return magazine.expendRounds(numToShoot);
    }

}
```

A.53 SSTAF/src/features/equipment/mil.devcom_dac.equipment.api/src/main/java/mil/devcom_dac/equipment/api/Item.java

```
package mil.devcom_dac.equipment.api;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.Getter;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import mil.sstaf.core.util.SSTAFException;
import java.util.Objects;
@SuperBuilder
@Jacksonized
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
public class Item {
    @Getter
    protected String name;
    protected Item(ItemBuilder<?, ?> b) {
        this.name = Objects.requireNonNullElse(b.name, this.getClass().getSimpleName());
    }
    public double getMass_kg() {
        throw new SSTAFException("getMass_kg must be overridden");
    };
}
```

A.54 SSTAF/src/features/equipment/mil.devcom_dac.equipment.api/src/main/java/mil/devcom_dac/equipment/api/Kit.java

```
package mil.devcom_dac.equipment.api;

import com.fasterxml.jackson.annotation.JsonIgnore;
import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import lombok.Builder;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import java.util.*;
@SuperBuilder
@Jacksonized
@JsonIgnoreProperties(value={"class"})
public class Kit implements SoldierKit {
    private List<Magazine> magazines;
    private List<Gun> guns;
    private List<Pack> packs;
    @Builder.Default
    @JsonIgnore
    private Map<String, List<Magazine>> magazineMap = null;
    @Builder.Default
    @JsonIgnore
    private Map<String, Gun> gunMap = null;
    @Builder.Default
    @JsonIgnore
    private Map<String, Pack> packMap = null;
    @Builder.Default
    @JsonIgnore
    private List<Item> allItems = null;
    public Kit(KitBuilder<?, ?> builder) {
        // 2022.04.21 : RAB : Packs, Guns and Magazines are created as
        //                    modifiable collections.
        this.gunMap = new HashMap<>();
        if (builder.guns != null) {
            for (Gun g : builder.guns) {
                this.gunMap.put(g.getName(), g);
            }
        }
        this.packMap = new HashMap<>();
        if (builder.packs != null) {
            for (Pack p : builder.packs) {
                this.packMap.put(p.getName(), p);
            }
        }
        this.magazineMap = new HashMap<>();
        if (builder.magazines != null) {
            for (var mag : builder.magazines) {
                List<Magazine> magList;
                if (this.magazineMap.containsKey(mag.getMagazineType())) {
                    magList = magazineMap.get(mag.getMagazineType());
                } else {
                    magList = new ArrayList<>();
                    magazineMap.put(mag.getMagazineType(), magList);
                }
                magList.add(mag);
            }
        }
        this.allItems = new ArrayList<>();
        for (var x : this.gunMap.entrySet()) {
            allItems.add(x.getValue());
        }
        for (var x : this.packMap.entrySet()) {
            allItems.add(x.getValue());
        }
```

```
        for (var es : this.magazineMap.entrySet()) {
            allItems.addAll(es.getValue());
        }
    }
    static private void checkNullGun(Gun gun) {
        if (gun == null) {
            throw new IllegalArgumentException("Gun must not be null.");
        }
    }
    static private void checkNullMagazine(Magazine magazine) {
        if (magazine == null) {
            throw new IllegalArgumentException("Magazine must not be null.");
        }
    }
    static private void checkNullPack(Pack pack) {
        if (pack == null) {
            throw new IllegalArgumentException("Pack must not be null.");
        }
    }
    public Item getItemByName(String name) {
        Objects.requireNonNull(name, "Item name was null");
        Item found = null;
        for (Item item : allItems) {
            if (name.equals(item.getName())) {
                found = item;
            }
        }
        return found;
    }
    /**
     * Provides the gun with the given name,if it exists
     *
     * @param gunName the name of the {@code Gun}
     * @return the Gun or null
     */
    @Override
    public Gun getGunByName(final String gunName) {
        return gunMap.get(Objects.requireNonNull(gunName, "Gun name was null"));
    }
    @Override
    public Map<String, Gun> getGuns() {
        return Collections.unmodifiableMap(gunMap);
    }
    @Override
    public Map<String, Pack> getPacks() {
        return Collections.unmodifiableMap(packMap);
    }
    @Override
    public List<Magazine> getMagazines(String magazineType) {
        return magazineMap.getOrDefault(magazineType, Collections.emptyList());
    }
    @Override
    public double getMass() {
        double mass = 0.0;
        for (Gun gun : gunMap.values()) {
            mass += gun.getMass_kg();
        }
        for (List<Magazine> magSets : magazineMap.values()) {
            for (Magazine mag : magSets) {
                mass += mag.getMass_kg();
            }
        }
        for (Pack pack : packMap.values()) {
            mass += pack.getMass_kg();
        }
        return mass;
    }
    @Override
```

```
    public boolean canReload(Gun gun) {
        return !getMagazines(gun.getMagazineType()).isEmpty();
    }
    @Override
    public boolean reload(Gun gun) {
        List<Magazine> mags = getMagazines(gun.getMagazineType());
        if (mags.isEmpty()) {
            return false;
        } else {
            Magazine mag = mags.remove(mags.size() - 1);
            gun.loadMagazine(mag);
            return true;
        }
    }
    @Override
    public void drop(Gun gun) {
        checkNullGun(gun);
        gunMap.remove(gun.getName());
    }
    @Override
    public void drop(Pack pack) {
        checkNullPack(pack);
        packMap.remove(pack.getName());
    }
    @Override
    public void add(Gun gun) {
        checkNullGun(gun);
        gunMap.put(gun.getName(), gun);
        allItems.add(gun);
    }
    @Override
    public void add(Pack pack) {
        checkNullPack(pack);
        packMap.put(pack.getName(), pack);
        allItems.add(pack);
    }
    @Override
    public void add(Magazine mag) {
        checkNullMagazine(mag);
        List<Magazine> magazines = this.magazineMap.getOrDefault(mag.getMagazineType(), new Array
List<>(10));
        magazines.add(mag);
        this.magazineMap.put(mag.getMagazineType(), magazines);
        allItems.add(mag);
    }
    @Override
    public void drop(Magazine mag) {
        checkNullMagazine(mag);
        if (this.magazineMap.containsKey(mag.getMagazineType())) {
            List<Magazine> mags = this.magazineMap.get(mag.getMagazineType());
            mags.remove(mag);
        }
    }
}
```

A.55 SSTAF/src/features/equipment/mil.devcom_dac.equipment.api/src/main/java/mil/devcom_dac/equipment/api/KitState.java

```
package mil.devcom_dac.equipment.api;
import lombok.Builder;
import mil.sstaf.core.state.StateProperty;
/**
 * State object for reporting to Telemetry
 */
@Builder
public class KitState {
    public static final String BK_KEY = "KitState";
    private final double kitMass;
    private final double gunMass;
    private final int numRoundsRemaining;
    @StateProperty(headerLabel = "kitMass")
    public double getKitMass() {
        return kitMass;
    }
    @StateProperty(headerLabel = "gunMass")
    public double getGunMass() {
        return gunMass;
    }
    @StateProperty(headerLabel = "numRoundsRemaining")
    public int getNumRoundsRemaining() {
        return numRoundsRemaining;
    }

}
```

A.56 SSTAF/src/features/equipment/mil.devcom_dac.equipment.api/src/main/java/mil/devcom_dac/equipment/api/Magazine.java

```
package mil.devcom_dac.equipment.api;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
@SuperBuilder
@Jacksonized
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
@EqualsAndHashCode(callSuper = true)
public class Magazine extends Item {
    @Getter
    private String magazineType;
    @Getter
    private double emptyMass_kg;
    @Getter
    private int capacity;
    @Getter
    private double perRoundMass_kg;
    @Getter
    private int currentLoad;
    private Magazine(MagazineBuilder<?,?> b) {
        super(b);
        if (b.magazineType == null) {
            // 2022.04.21 : RAB : Default is to generate a fully loaded 5.56
            this.capacity = b.capacity == 0 ? 30 : b.capacity;
            this.currentLoad = b.currentLoad == 0 ? this.capacity : b.currentLoad;
            this.magazineType = "5.56mm STANAG";
            this.name = "5.56mm STANAG 30rd";
            this.perRoundMass_kg = 0.01231;
            this.emptyMass_kg = 0.1207;
        } else {
            this.magazineType = b.magazineType;
            this.emptyMass_kg = b.emptyMass_kg;
            this.perRoundMass_kg = b.perRoundMass_kg;
            this.capacity = b.capacity;
            if (b.currentLoad >= 0 && b.currentLoad <= capacity) {
                this.currentLoad = b.currentLoad;
            } else {
                this.currentLoad = capacity;
            }
        }
    }
    public double getMass_kg() {
        return emptyMass_kg + currentLoad * perRoundMass_kg;
    }
    /**
     * Removes
     *
     * @param numToExpend the desired number of rounds to expend
     * @return the number of rounds expended
     */
    public int expendRounds(final int numToExpend) {
        if (numToExpend < 0) {
            throw new IllegalArgumentException("numToExpend must be non-negative");
        }
        int shot = Math.min(numToExpend, currentLoad);
        currentLoad -= shot;
        return shot;
    }
```

```
    public boolean isEmpty() {
        return currentLoad == 0;
    }
}
```

A.57 SSTAF/src/features/equipment/mil.devcom_dac.equipment.api/src/main/java/mil/devcom_dac/equipment/api/Pack.java

```
package mil.devcom_dac.equipment.api;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
@SuperBuilder
@Jacksonized
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
@EqualsAndHashCode(callSuper = true)
public class Pack extends Item {
    @Getter
    private final double mass_kg;
    protected Pack(PackBuilder<?,?> b) {
        super(b);
        this.mass_kg = b.mass_kg;
    }
}
```

A.58 SSTAF/src/features/equipment/mil.devcom_dac.equipment.api/src/main/java/mil/devcom_dac/equipment/api/SoldierKit.java

```
package mil.devcom_dac.equipment.api;
import java.util.List;
import java.util.Map;
public interface SoldierKit {
    Item getItemByName(String itemName);
    Gun getGunByName(String gunName);
    Map<String, Gun> getGuns();
    Map<String, Pack> getPacks();
    List<Magazine> getMagazines(String magazineType);
    double getMass();
    boolean canReload(Gun gun);
    boolean reload(Gun gun);
    void drop(Gun gun);
    void drop(Pack pack);
    void add(Gun gun);
    void add(Pack pack);
    void add(Magazine mag);
    void drop(Magazine mag);
}
```

A.59 SSTAF/src/features/equipment/mil.devcom_dac.equipment.api/src/main/java/module-info.java

```
module mil.devcom_dac.equipment.api {
    exports mil.devcom_dac.equipment.api;
    requires mil.sstaf.core;

    requires org.slf4j;
    opens mil.devcom_dac.equipment.api to com.fasterxml.jackson.databind;
}
```

A.60 SSTAF/src/features/equipment/mil.devcom_dac.equipment.api/src/test/java/mil/devcom_dac/equipment/api/EquipmentConfigurationTest.java

```
package mil.devcom_dac.equipment.api;
import mil.sstaf.core.json.JsonLoader;
import org.junit.jupiter.api.Test;
import java.nio.file.Path;
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertNotNull;
public class EquipmentConfigurationTest {
    @Test
    void canCreateDefault() {
        EquipmentConfiguration ec = EquipmentConfiguration.builder().build();
        Kit kit = ec.getKit();
        assertNotNull(kit);
        assertEquals(0.0, kit.getMass());
    }
    @Test
    void goodFileWorks() {
        String kitFile = "src/test/resources/TestConfig.json";
        EquipmentConfiguration equipmentConfiguration =
                new JsonLoader().load(Path.of(kitFile), EquipmentConfiguration.class);
        assertNotNull(equipmentConfiguration);
        Kit kit = equipmentConfiguration.getKit();
        assertNotNull(kit);
        assertEquals(4, kit.getMagazines("5.56mm STANAG").size());
        assertEquals(1, kit.getGuns().size());
        assertEquals(1, kit.getPacks().size());
    }
}
```

A.61 SSTAF/src/features/equipment/mil.devcom_dac.equipment.api/src/test/java/mil/devcom_dac/equipment/api/GunTest.java

```
package mil.devcom_dac.equipment.api;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;
class GunTest {
    @Test
    void canCreateDefault() {
        var gun = Gun.builder().build();
        Assertions.assertNotNull(gun);
        Assertions.assertEquals("Gun", gun.getName());
    }
    @Test
    void builderWorks() {
        final String NAME = "M4";
        final double EMPTY_MASS = 3.5;
        var builder = Gun.builder();
        builder.name(NAME);
        builder.emptyMass_kg(EMPTY_MASS);
        var gun = builder.build();
        Assertions.assertEquals(NAME, gun.getName());
        Assertions.assertEquals(EMPTY_MASS, gun.getEmptyMass_kg());
        Assertions.assertEquals(EMPTY_MASS, gun.getMass_kg());
    }
    @Test
    void magazineAndAmmoManagementWorks() {
        final String NAME = "M4";
        final double EMPTY_MASS = 3.5;
        var builder = Gun.builder();
        builder.name(NAME);
        builder.emptyMass_kg(EMPTY_MASS);
        var gun = builder.build();
        var mb1 = Magazine.builder();
        mb1.name("mag1");
        var mag1 = mb1.build();
        var mb2 = Magazine.builder();
        mb2.name("mag2");
        var mag2 = mb1.build();
        Assertions.assertEquals(NAME, gun.getName());
        Assertions.assertEquals(EMPTY_MASS, gun.getEmptyMass_kg());
        Assertions.assertEquals(EMPTY_MASS, gun.getMass_kg());
        var massMag1 = mag1.getMass_kg();
        var massMag2 = mag2.getMass_kg();
        gun.loadMagazine(mag1);
        Assertions.assertEquals(EMPTY_MASS + massMag1, gun.getMass_kg());
        while (gun.canShoot()) {
            int shot = gun.shoot(1);
            Assertions.assertTrue(shot > 0);
        }
        Assertions.assertEquals(EMPTY_MASS + mag1.getMass_kg(), gun.getMass_kg());
        Assertions.assertEquals(EMPTY_MASS + mag1.getEmptyMass_kg(), gun.getMass_kg());
        gun.loadMagazine(mag2);
        Assertions.assertEquals(EMPTY_MASS + massMag2, gun.getMass_kg());
        while (gun.canShoot()) {
            int shot = gun.shoot(3);
            Assertions.assertTrue(shot > 0);
        }
        Assertions.assertEquals(EMPTY_MASS + mag2.getMass_kg(), gun.getMass_kg());
        Assertions.assertEquals(EMPTY_MASS + mag2.getEmptyMass_kg(), gun.getMass_kg());
        gun.dropMagazine();
        Assertions.assertEquals(EMPTY_MASS, gun.getEmptyMass_kg());
        Assertions.assertEquals(EMPTY_MASS, gun.getMass_kg());
```

```
    }
}
```

A.62 SSTAF/src/features/equipment/mil.devcom_dac.equipment.api/src/test/java/mil/devcom_dac/equipment/api/KitTest.java

```
package mil.devcom_dac.equipment.api;
import com.fasterxml.jackson.annotation.JsonIgnore;
import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import mil.sstaf.core.json.JsonLoader;
import org.junit.jupiter.api.Test;
import java.nio.file.Path;
import java.util.Collections;
import java.util.Map;
import static org.junit.jupiter.api.Assertions.*;

class KitTest {
    @Test
    void canCreateDefault() {
        Kit kit = Kit.builder().build();
        assertNotNull(kit);
        assertEquals(0.0, kit.getMass());
    }
    @Test
    void builderWorks() {
        Kit kit = Kit.builder().build();
        var mag1 = Magazine.builder().build();
        var mag2 = Magazine.builder().build();
        var gun1 = Gun.builder().build();
        var gun2 = Gun.builder().name("M17").build();
        var pack1 = Pack.builder().build();
        var pack2 = Pack.builder().name("Fanny").build();
        kit.add(mag1);
        kit.add(mag2);
        kit.add(gun1);
        kit.add(gun2);
        kit.add(pack1);
        kit.add(pack2);
        assertEquals(2, kit.getMagazines(mag1.getMagazineType()).size());
        assertEquals(0, kit.getMagazines("M14 20rd").size());
        assertEquals(2, kit.getGuns().size());
        assertEquals(2, kit.getPacks().size());
    }
    @Test
    void addAndDropWork() {
        var kit = Kit.builder().build();
        var gun = Gun.builder().build();
        var mag = Magazine.builder().build();
        var pack = Pack.builder().build();
        kit.add(gun);
        assertEquals(gun.getMass_kg(), kit.getMass());
        kit.add(pack);
        assertEquals(gun.getMass_kg() + pack.getMass_kg(), kit.getMass());
        kit.add(mag);
        assertEquals(gun.getMass_kg() + pack.getMass_kg() + mag.getMass_kg(),
                kit.getMass());
        kit.drop(mag);
        assertEquals(gun.getMass_kg() + pack.getMass_kg(),
                kit.getMass());
        kit.drop(pack);
        assertEquals(gun.getMass_kg(), kit.getMass());
        kit.drop(gun);
        assertEquals(0, kit.getMass());
    }
    @Test
    void nullsCauseThrows() {
        var kit = Kit.builder().build();
        assertThrows(IllegalArgumentException.class,
                () -> kit.add((Gun) null));
```

```
        assertThrows(IllegalArgumentException.class,
                () -> kit.add((Magazine) null));
        assertThrows(IllegalArgumentException.class,
                () -> kit.add((Pack) null));
        assertThrows(IllegalArgumentException.class,
                () -> kit.drop((Gun) null));
        assertThrows(IllegalArgumentException.class,
                () -> kit.drop((Magazine) null));
        assertThrows(IllegalArgumentException.class,
                () -> kit.drop((Pack) null));
    }
    @Test
    void reloadLogicWorks() {
        var bldr = Magazine.builder();
        bldr.magazineType("5.56mm STANAG");

        var mag4_1 = bldr.build();
        var mag4_2 = bldr.build();
        var mag4_3 = bldr.build();
        var mag4_4 = bldr.build();
        var bldr14 = Magazine.builder();
        bldr14.magazineType("7.62x51 M14");

        var mag14_1 = bldr14.build();
        var mag14_2 = bldr14.build();
        var mag14_3 = bldr14.build();
        var mag14_4 = bldr14.build();
        var bldr_m4 = Gun.builder();
        bldr_m4.name("M4");
        bldr_m4.magazineType("5.56mm STANAG");
        var m4 = bldr_m4.build();
        var bldr_m14 = Gun.builder();
        bldr_m14.name("M14");
        bldr_m14.magazineType("7.62x51 M14");
        var m14 = bldr_m14.build();
        var bldr_m41 = Gun.builder();
        bldr_m41.name("M41");
        bldr_m41.magazineType("10mm Caseless");
        var m41 = bldr_m41.build();

        var kit = Kit.builder().build();
        kit.add(mag4_1);
        kit.add(mag4_2);
        kit.add(mag4_3);
        kit.add(mag4_4);
        kit.add(mag14_1);
        kit.add(mag14_2);
        kit.add(mag14_3);
        kit.add(mag14_4);
        assertTrue(kit.canReload(m4));
        assertTrue(kit.canReload(m14));
        assertFalse(kit.canReload(m41));
        kit.drop(mag14_1);
        kit.drop(mag14_2);
        kit.drop(mag14_3);
        kit.drop(mag14_4);
        assertFalse(kit.canReload(m14));
        assertTrue(kit.reload(m4));
        assertFalse(kit.reload(m14));
        assertFalse(kit.reload(m41));
    }

    @Test
    void aFullScenarioWorks() {
        var bldr = Magazine.builder();
        bldr.magazineType("5.56mm STANAG");
        var mag1 = bldr.build();
        var mag2 = bldr.build();
```

```
        var mag3 = bldr.build();
        var mag4 = bldr.build();
        var mag5 = bldr.build();
        var mag6 = bldr.build();
        var mag7 = bldr.build();
        var gbldr = Gun.builder();
        gbldr.name("M4");
        gbldr.magazineType("5.56mm STANAG");
        var gun = gbldr.build();
        var pb = Pack.builder();
        pb.mass_kg(30);
        var pack = pb.build();
        assertEquals(30.0, pack.getMass_kg());
        var kit = Kit.builder().build();
        kit.add(gun);
        kit.add(pack);
        kit.add(mag1);
        kit.add(mag2);
        kit.add(mag3);
        kit.add(mag4);
        kit.add(mag5);
        kit.add(mag6);
        kit.add(mag7);
        Item[] allItems = {mag1, mag2, mag3, mag4, mag5, mag6, mag7, gun, pack};
        double totalMass = 0.0;
        for (Item item : allItems) {
            totalMass += item.getMass_kg();
        }
        assertEquals(totalMass, kit.getMass());
        //
        // Load the gun.
        //
        double expectedMass = totalMass;
        int magsDumped = 0;
        while (kit.canReload(gun)) {
            kit.reload(gun);
            while (gun.canShoot()) {
                gun.shoot(1);
                expectedMass = expectedMass - mag1.getPerRoundMass_kg();
                assertEquals(expectedMass, kit.getMass(), 0.000001);
            }
            assertEquals(gun.getEmptyMass_kg() + mag1.getEmptyMass_kg(), gun.getMass_kg());
            gun.dropMagazine();
            ++magsDumped;
            expectedMass -= mag1.getEmptyMass_kg();
            assertEquals(gun.getEmptyMass_kg(), gun.getMass_kg());
            assertEquals(expectedMass, kit.getMass(), 0.000001);
        }
        assertEquals(pack.getMass_kg() + gun.getEmptyMass_kg(), kit.getMass(), 0.000001);
        assertEquals(7, magsDumped);
    }
    @Test
    void emptySpecificationProduceEmptyKit() {
        Map<String, Object> map = Collections.emptyMap();
        EquipmentConfiguration equipmentConfiguration =
                EquipmentConfiguration.builder().build();
        Kit kit = equipmentConfiguration.getKit();
        assertNotNull(kit);
        assertEquals(0.0, kit.getMass());
    }

    @Test
    void goodFileWorks() {
        String kitFile = "src/test/resources/TestKit.json";
        Kit kit = new JsonLoader().load(Path.of(kitFile), Kit.class);
        assertEquals(4, kit.getMagazines("5.56mm STANAG").size());
        assertEquals(1, kit.getGuns().size());
        assertEquals(1, kit.getPacks().size());
```

```
    }
}
```

A.63 SSTAF/src/features/equipment/mil.devcom_dac.equipment.api/src/test/java/mil/devcom_dac/equipment/api/MagazineTest.java

```
package mil.devcom_dac.equipment.api;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;
class MagazineTest {
    @Test
    void canCreateDefault() {
        var magazine = Magazine.builder().build();
        Assertions.assertNotNull(magazine);
        Assertions.assertEquals(30, magazine.getCapacity());
        Assertions.assertEquals(30, magazine.getCurrentLoad());
        Assertions.assertEquals(0.01231, magazine.getPerRoundMass_kg());
        Assertions.assertEquals(0.1207, magazine.getEmptyMass_kg());
        Assertions.assertEquals(0.490, magazine.getMass_kg());
        Assertions.assertEquals("5.56mm STANAG 30rd", magazine.getName());
        Assertions.assertEquals("5.56mm STANAG", magazine.getMagazineType());
    }
    @Test
    void builderWorks() {
        final double EMPTY_MASS = 20.0;
        final double ROUND_MASS = 4.0;
        final int CAPACITY = 4;
        final int LOAD = 3;
        final String NAME = "Fred";
        final String GUN_NAME = "Phaser";
        var builder = Magazine.builder();
        builder.perRoundMass_kg(ROUND_MASS);
        builder.name(NAME);
        builder.emptyMass_kg(EMPTY_MASS);
        builder.magazineType(GUN_NAME);
        builder.capacity(CAPACITY);
        builder.currentLoad(LOAD);
        var magazine = builder.build();
        Assertions.assertEquals(CAPACITY, magazine.getCapacity());
        Assertions.assertEquals(LOAD, magazine.getCurrentLoad());
        Assertions.assertEquals(ROUND_MASS, magazine.getPerRoundMass_kg());
        Assertions.assertEquals(EMPTY_MASS, magazine.getEmptyMass_kg());
        Assertions.assertEquals(EMPTY_MASS + ROUND_MASS * LOAD, magazine.getMass_kg());
        Assertions.assertEquals(NAME, magazine.getName());
        Assertions.assertEquals(GUN_NAME, magazine.getMagazineType());
    }
    @Test
    void singleRoundExpenditureWorks() {
        var magazine = Magazine.builder().build();
        int lastRoundCount = magazine.getCurrentLoad();
        int initialCount = lastRoundCount;
        int totalFired = 0;
        while (magazine.expendRounds(1) > 0) {
            ++totalFired;
            Assertions.assertEquals(lastRoundCount - 1, magazine.getCurrentLoad());
            lastRoundCount = magazine.getCurrentLoad();
        }
        Assertions.assertEquals(magazine.getEmptyMass_kg(), magazine.getMass_kg());
        Assertions.assertEquals(initialCount, totalFired);
    }
    @Test
    void multiRoundExpenditureAndIsEmptyWorks() {
        var magazine = Magazine.builder().build();
        int lastRoundCount = magazine.getCurrentLoad();
        final int initialCapacity = lastRoundCount;
        int totalFired = 0;
```

```
        final int NUM_TO_FIRE = 3;
        while (!magazine.isEmpty()) {
            magazine.expendRounds(NUM_TO_FIRE);
            totalFired += NUM_TO_FIRE;
            Assertions.assertEquals(lastRoundCount - NUM_TO_FIRE, magazine.getCurrentLoad());
            lastRoundCount = magazine.getCurrentLoad();
        }
        Assertions.assertEquals(magazine.getEmptyMass_kg(), magazine.getMass_kg());
        Assertions.assertEquals(initialCapacity, totalFired);
    }
    @Test
    void negativeExpenditureThrows() {
        var b = Magazine.builder();
        b.capacity(60);
        var mag1 = b.build();
        var mag2 = Magazine.builder()
                .magazineType(mag1.getMagazineType())
                .currentLoad(mag1.getCurrentLoad())
                .emptyMass_kg(mag1.getMass_kg())
                .build();
        Assertions.assertThrows(IllegalArgumentException.class, () -> mag1.expendRounds(-121));
    }
}
```

A.64 SSTAF/src/features/equipment/mil.devcom_dac.equipment.api/src/test/java/mil/devcom_dac/equipment/api/PackTest.java

```
package mil.devcom_dac.equipment.api;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;
class PackTest {
    @Test
    void canCreateDefault() {
        var pack = Pack.builder().build();
        Assertions.assertNotNull(pack);
        Assertions.assertEquals("Pack", pack.getName());
    }
    @Test
    void builderWorks() {
        final String NAME = "Backpack";
        final double MASS = 30.7;
        var pb = Pack.builder();
        pb.mass_kg(MASS);
        pb.name(NAME);
        var pack = pb.build();
        Assertions.assertEquals(NAME, pack.getName());
        Assertions.assertEquals(MASS, pack.getMass_kg());
    }
}
```

A.65 SSTAF/src/features/equipment/mil.devcom_dac.equipment.api/src/test/resources/FannyPack.json

```
{
  "class" : "mil.devcom_dac.equipment.api.Pack",
  "name": "Fanny Pack",
  "mass_kg": 1.5
}
```

A.66 SSTAF/src/features/equipment/mil.devcom_dac.equipment.api/src/test/resources/M16A1.json

```
{
  "class" : "mil.devcom_dac.equipment.api.Gun",
  "name": "M16A1",
  "magazineType": "5.56mm STANAG",
  "emptyMass_kg": 5.5
}
```

A.67 SSTAF/src/features/equipment/mil.devcom_dac.equipment.api/src/test/resources/STANAGMagazine.json

```
{
  "class" : "mil.devcom_dac.equipment.api.Magazine",
  "name": "5.56mm STANAG 30rd",
  "magazineType": "5.56mm STANAG",
  "capacity": 30,
  "currentLoad" : 30,
  "perRoundMass_kg": 0.01231,
  "emptyMass_kg": 0.1207
}
```

A.68 SSTAF/src/features/equipment/mil.devcom_dac.equipment.api/src/test/resources/TestConfig.json

```
{
  "class" : "mil.devcom_dac.equipment.api.EquipmentConfiguration",
  "kit" : "TestKit.json"
}
```

A.69 SSTAF/src/features/equipment/mil.devcom_dac.equipment.api/src/test/resources/TestKit.json

```
{
  "class" : "mil.devcom_dac.equipment.api.Kit",
  "guns": [
    "M16A1.json"
  ],
  "magazines": [
    "STANAGMagazine.json",
    "STANAGMagazine.json",
    "STANAGMagazine.json",
    "STANAGMagazine.json"
  ],
  "packs": [
    "FannyPack.json"
  ]
}
```

A.70 SSTAF/src/features/equipment/mil.devcom_dac.equipment.api/src/test/resources/log4j2-test.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="INFO">
    <Appenders>
        <Console name="Console" target="SYSTEM_OUT">
            <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"/>
        </Console>
    </Appenders>
    <Loggers>
        <Root level="debug">
            <AppenderRef ref="Console"/>
        </Root>
    </Loggers>
</Configuration>
```

A.71 SSTAF/src/features/equipment/mil.devcom_dac.equipment.handler/build.gradle

```
dependencies {
    implementation project(':framework:mil.sstaf.core')
    implementation project(':features:equipment:mil.devcom_dac.equipment.messages')
    implementation project(':features:equipment:mil.devcom_dac.equipment.api')
}
```

A.72 SSTAF/src/features/equipment/mil.devcom_dac.equipment.handler/src/main/java/mil/devcom_dac/equipment/handler/EquipmentHandler.java

```
package mil.devcom_dac.equipment.handler;
import mil.devcom_dac.equipment.api.*;
import mil.devcom_dac.equipment.messages.*;
import mil.sstaf.core.entity.Address;
import mil.sstaf.core.entity.Message;
import mil.sstaf.core.features.BaseHandler;
import mil.sstaf.core.features.FeatureConfiguration;
import mil.sstaf.core.features.HandlerContent;
import mil.sstaf.core.features.ProcessingResult;
import mil.sstaf.core.util.SSTAFException;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import java.util.*;
public class EquipmentHandler extends BaseHandler implements EquipmentManagement {
    public static final String FEATURE_NAME = "Kit Manager";
    public static final int MAJOR_VERSION = 0;
    public static final int MINOR_VERSION = 1;
    public static final int PATCH_VERSION = 0;
    private static final Logger logger = LoggerFactory.getLogger(EquipmentHandler.class);
    private Kit kit;
    private Gun currentGun;

    public EquipmentHandler() {
        super(FEATURE_NAME, MAJOR_VERSION, MINOR_VERSION, PATCH_VERSION,
                true, "Handler for military equipment loading");
    }
    @Override
    public List<Class<? extends HandlerContent>> contentHandled() {
        return List.of();
    }
    @Override
    public void init() {
        super.init();
    }
    /**
     * {@inheritDoc}
     */
    @Override
    public void configure(FeatureConfiguration configuration) {
        if (logger.isTraceEnabled()) {
            logger.trace("{}:{} - Configuring with {}", ownerHandle.getPath(), featureName, confi
guration);
        }
        super.configure(configuration);
        if (configuration instanceof EquipmentConfiguration) {
            EquipmentConfiguration ec = (EquipmentConfiguration) configuration;
            kit = ec.getKit();
            if (ec.getCurrentGun() != null) {
                currentGun = kit.getGunByName(ec.getCurrentGun());
            } else {
                currentGun = kit.getGuns().values().iterator().next();
            }
        }
    }
        /**
     * {@inheritDoc}
     */
    @Override
    public Item getItemByName(String itemName) {
        Objects.requireNonNull(itemName, "Item name");
```

```
        return kit.getItemByName(itemName);
    }
    private void setGun(String gunName) {
         Gun gun = getGunByName(gunName);
        if (gun == null) {
            throw new SSTAFException("Gun " + gunName + " was not found");
        } else {
            currentGun = gun;
        }
    }
@Override
    public ProcessingResult process(HandlerContent arg, long scheduledTime_ms, long currentTime_m
s, Address from, long id, Address respondTo) {
        ProcessingResult rv;
        if (arg instanceof Shoot) {
            Shoot shootMessage = (Shoot) arg;
            if (shootMessage.getGun() != null) {
                setGun(shootMessage.getGun());
            }
            int numShot = currentGun.shoot(shootMessage.getNumToShoot());
            int remaining = currentGun.getMagazine().getCurrentLoad();
            GunState response = GunState.builder()
                    .numberShot(numShot)
                    .roundsInCurrentGun(remaining)
                    .currentGun(currentGun.getName())
                    .build();
            Message m = buildNormalResponse(response, id, respondTo);
            rv = ProcessingResult.of(m);
        } else if (arg instanceof GetInventory) {
            Inventory ir = buildInventory();
            Message m = buildNormalResponse(ir, id, respondTo);
            rv = ProcessingResult.of(m);
        } else if (arg instanceof Reload) {
            Reload rm = (Reload) arg;
            if (rm.getGun() != null) {
                setGun(rm.getGun());
            }
            reload(currentGun);
            GunState gs = GunState.builder()
                    .currentGun(currentGun.getName())
                    .roundsInCurrentGun(currentGun.getMagazine().getCurrentLoad())
                    .build();
            Message m = buildNormalResponse(gs, id, respondTo);
            rv = ProcessingResult.of(m);
        } else if (arg instanceof SetGun) {
            SetGun sg = (SetGun)arg;
            setGun(sg.getGun());
            GunState gs = GunState.builder()
                    .currentGun(currentGun.getName())
                    .roundsInCurrentGun(currentGun.getMagazine().getCurrentLoad())
                    .build();
            Message m = buildNormalResponse(gs, id, respondTo);
            rv = ProcessingResult.of(m);
        } else {
            rv = buildUnsupportedMessageResponse(arg, id, respondTo, new Throwable());
        }
        return rv;
}
    private Inventory buildInventory() {
        var builder = Inventory.builder();
        if (currentGun == null) {
            builder.currentGun("None");
            builder.roundsInCurrentGun(0);
        } else {
            builder.currentGun(currentGun.getName());
            builder.roundsInCurrentGun(currentGun.getMagazine().getCurrentLoad());
        }
        Map<String, Integer> mpt = new TreeMap<>();
```

```
        Map<String, Integer> rpt = new TreeMap<>();
        Set<String> magTypes = new TreeSet<>();
        Map<String, Double> guns = new TreeMap<>();
        for (Gun gun : kit.getGuns().values()) {
            guns.put(gun.getName(), getMass());
            magTypes.add(gun.getMagazineType());
        }
        builder.guns(guns);
        for (String type : magTypes) {
            List<Magazine> magazines = kit.getMagazines(type);
            int nr = 0;
            for (Magazine mag : magazines) {
                nr += mag.getCurrentLoad();
            }
            mpt.put(type, magazines.size());
            rpt.put(type, nr);
        }
        builder.roundsPerType(rpt);
        builder.magazinesPerType(mpt);
        Map<String, Double> packs = new TreeMap<>();
        for (Pack pack : kit.getPacks().values()) {
            packs.put(pack.getName(), pack.getMass_kg());
        }
        builder.packs(packs);
        builder.totalCarriedMass(kit.getMass());
        return builder.build();
    }
    @Override
    public Gun getGunByName(final String gunName) {
        Objects.requireNonNull(gunName, "gun name");
        return kit.getGunByName(gunName);
    }
    @Override
    public Map<String, Gun> getGuns() {
        return kit.getGuns();
    }
    @Override
    public Map<String, Pack> getPacks() {
        return kit.getPacks();
    }
    @Override
    public List<Magazine> getMagazines(String magazineType) {
        return kit.getMagazines(magazineType);
    }
    @Override
    public double getMass() {
        return kit.getMass();
    }
    @Override
    public boolean canReload(Gun gun) {
        return kit.canReload(gun);
    }
    @Override
    public boolean reload(Gun gun) {
        return kit.reload(gun);
    }
    @Override
    public void drop(Gun gun) {
        kit.drop(gun);
    }
    @Override
    public void drop(Pack pack) {
        kit.drop(pack);
    }
    @Override
    public void add(Gun gun) {
        kit.add(gun);
    }
```

```
    @Override
    public void add(Pack pack) {
        kit.add(pack);
    }
    @Override
    public void add(Magazine mag) {
        kit.add(mag);
    }
    @Override
    public void drop(Magazine mag) {
        kit.drop(mag);
    }
}
```

A.73 SSTAF/src/features/equipment/mil.devcom_dac.equipment.handler/src/main/java/module-info.java

```
import mil.devcom_dac.equipment.api.EquipmentManagement;
import mil.devcom_dac.equipment.api.SoldierKit;
import mil.devcom_dac.equipment.handler.EquipmentHandler;
import mil.sstaf.core.features.Feature;
module mil.devcom_dac.equipment.handler {
    exports mil.devcom_dac.equipment.handler;
    requires mil.sstaf.core;
    requires mil.devcom_dac.equipment.messages;
    requires mil.devcom_dac.equipment.api;
    requires org.slf4j;
    provides EquipmentManagement with EquipmentHandler;
    provides SoldierKit with EquipmentHandler;
    provides Feature with EquipmentHandler;
}
```

A.74 SSTAF/src/features/equipment/mil.devcom_dac.equipment.messages/build.gradle

```
dependencies {
    implementation project(':framework:mil.sstaf.core')
    implementation 'org.projectlombok:lombok:1.18.20'
}
```

A.75 SSTAF/src/features/equipment/mil.devcom_dac.equipment.messages/src/main/java/mil/devcom_dac/equipment/messages/GetInventory.java

```
package mil.devcom_dac.equipment.messages;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import mil.sstaf.core.features.HandlerContent;
/**
 * Message to request the current inventory for the Soldier
 */
@SuperBuilder
@Jacksonized
public class GetInventory extends HandlerContent {
}
```

A.76 SSTAF/src/features/equipment/mil.devcom_dac.equipment.messages/src/main/java/mil/devcom_dac/equipment/messages/GunState.java

```
package mil.devcom_dac.equipment.messages;
import lombok.Getter;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import mil.sstaf.core.features.HandlerContent;
@SuperBuilder
@Jacksonized
public class GunState extends HandlerContent {
    @Getter
    private String currentGun;
    @Getter
    private int numberShot;
    @Getter
    private int roundsInCurrentGun;
}
```

A.77 SSTAF/src/features/equipment/mil.devcom_dac.equipment.messages/src/main/java/mil/devcom_dac/equipment/messages/Inventory.java

```
package mil.devcom_dac.equipment.messages;
import lombok.Getter;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import mil.sstaf.core.features.HandlerContent;
import java.util.Map;
/**
 * Response for GetInventory
 */
@SuperBuilder
@Jacksonized
public class Inventory extends HandlerContent{
    @Getter
    private String currentGun;
    @Getter
    private int roundsInCurrentGun;
    @Getter
    private double totalCarriedMass;
    @Getter
    private Map<String, Double> guns;
    @Getter
    private Map<String, Integer> magazinesPerType;
    @Getter
    private Map<String, Integer> roundsPerType;
    @Getter
    private Map<String, Double> packs;
}
```

A.78 SSTAF/src/features/equipment/mil.devcom_dac.equipment.messages/src/main/java/mil/devcom_dac/equipment/messages/Reload.java

```
package mil.devcom_dac.equipment.messages;
import lombok.Getter;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import mil.sstaf.core.features.HandlerContent;
@SuperBuilder
@Jacksonized
public class Reload extends HandlerContent {
    @Getter
    String gun;
}
```

A.79 SSTAF/src/features/equipment/mil.devcom_dac.equipment.messages/src/main/java/mil/devcom_dac/equipment/messages/SetGun.java

```
package mil.devcom_dac.equipment.messages;
import lombok.Getter;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import mil.sstaf.core.features.HandlerContent;
@SuperBuilder
@Jacksonized
public class SetGun extends HandlerContent {
    @Getter
    private String gun;
}
```

A.80 SSTAF/src/features/equipment/mil.devcom_dac.equipment.messages/src/main/java/mil/devcom_dac/equipment/messages/Shoot.java

```
package mil.devcom_dac.equipment.messages;
import lombok.Getter;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import mil.sstaf.core.features.HandlerContent;
@SuperBuilder
@Jacksonized
public class Shoot extends HandlerContent {
    @Getter
    private String gun;
    @Getter
    private int numToShoot;
}
```

A.81 SSTAF/src/features/equipment/mil.devcom_dac.equipment.messages/src/main/java/module-info.java

```
module mil.devcom_dac.equipment.messages {
    exports mil.devcom_dac.equipment.messages;
    requires transitive mil.sstaf.core;
    requires lombok;
    requires com.fasterxml.jackson.databind;
    opens mil.devcom_dac.equipment.messages to com.fasterxml.jackson.databind;
}
```

A.82 SSTAF/src/features/simplePhysiologyAgent/mil.sstaf.physiology.agent/build.gradle

```
ext.moduleName = 'mil.sstaf.physiology.agent'
dependencies {
    implementation project(':framework:mil.sstaf.core')
    implementation project(':features:support:mil.sstaf.blackboard.api');
    implementation group: 'org.slf4j', name: 'slf4j-api', version: '1.7.32'
    implementation project(':features:ansurHandler:mil.devcom_sc.ansur.messages')
    implementation project(':features:ansurHandler:mil.devcom_sc.ansur.api')
    implementation project(':features:simplePhysiologyAgent:mil.sstaf.physiology.api')
    implementation project(':features:simplePhysiologyAgent:mil.sstaf.physiology.models.api')
    testRuntimeOnly project(':features:support:mil.sstaf.blackboard.inmem')
    testRuntimeOnly project(':features:ansurHandler:mil.devcom_sc.ansur.handler')
    testRuntimeOnly project(':features:simplePhysiologyAgent:mil.sstaf.physiology.models.musculat
ure')
    testRuntimeOnly project(':features:simplePhysiologyAgent:mil.sstaf.physiology.models.energy')
    testRuntimeOnly project(':features:simplePhysiologyAgent:mil.sstaf.physiology.models.vision')
    testRuntimeOnly project(':features:simplePhysiologyAgent:mil.sstaf.physiology.models.cardiova
scular')
    testRuntimeOnly project(':features:simplePhysiologyAgent:mil.sstaf.physiology.models.cognitio
n')
    testRuntimeOnly project(':features:simplePhysiologyAgent:mil.sstaf.physiology.models.respirat
ion')
    testRuntimeOnly project(':features:simplePhysiologyAgent:mil.sstaf.physiology.models.hydratio
n')
    //testRuntimeOnly project(':mil.sstaf.anthropometry.simple')
}
```

A.83 SSTAF/src/features/simplePhysiologyAgent/mil.sstaf.physiology.agent/src/main/java/mil/sstaf/physiology/agent/PhysiologyAgent.java

```
package mil.sstaf.physiology.agent;
import mil.sstaf.blackboard.api.Blackboard;
import mil.sstaf.core.entity.Address;
import mil.sstaf.core.features.BaseAgent;
import mil.sstaf.core.features.HandlerContent;
import mil.sstaf.core.features.ProcessingResult;
import mil.sstaf.core.features.Requires;
import mil.sstaf.physiology.api.PhysiologyManagement;
import mil.sstaf.physiology.api.PhysiologyState;
import mil.sstaf.physiology.models.api.*;
import org.slf4j.LoggerFactory;
import org.slf4j.Logger;
import java.util.List;
import java.util.Objects;
public class PhysiologyAgent extends BaseAgent implements PhysiologyManagement {
    public static final String FEATURE_NAME = "Physiology Agent";
    public static final int MAJOR_VERSION = 0;
    public static final int MINOR_VERSION = 1;
    public static final int PATCH_VERSION = 0;
    private static final Logger logger = LoggerFactory.getLogger(PhysiologyAgent.class);
    @Requires
    Blackboard blackboard;
    @Requires()
    CognitionModel brainModel;
    @Requires()
    CardiovascularModel heartModel;
    @Requires()
    RespirationModel respirationModel;
    @Requires()
    EnergyModel energyModel;
    @Requires()
    HydrationModel hydrationModel;
    @Requires()
    MusculatureModel musculatureModel;
    @Requires()
    VisionModel visionModel;
    public PhysiologyAgent() {
        super(FEATURE_NAME, MAJOR_VERSION, MINOR_VERSION, PATCH_VERSION,
                false, "Unified physiology");
    }
    @Override
    public void init() {
        super.init();
        logger.trace("Initializing PhysiologyAgent");
        Objects.requireNonNull(blackboard);
        Objects.requireNonNull(brainModel);
        Objects.requireNonNull(heartModel);
        Objects.requireNonNull(respirationModel);
        Objects.requireNonNull(hydrationModel);
        Objects.requireNonNull(musculatureModel);
        Objects.requireNonNull(energyModel);
        Objects.requireNonNull(visionModel);
    }
    @Override
    public PhysiologyState getState(long time_ms) {
        PhysiologyState.PhysiologyStateBuilder<?,?> builder = PhysiologyState.builder();
        builder.owner(ownerHandle);
        builder.timestamp_ms(time_ms);
        builder.cardiovascularMetrics(heartModel.evaluate(time_ms));
        builder.cognitionMetrics(brainModel.evaluate(time_ms));
```

```
        builder.energyMetrics(energyModel.evaluate(time_ms));
        builder.hydrationMetrics(hydrationModel.evaluate(time_ms));
        builder.musculatureMetrics(musculatureModel.evaluate(time_ms));
        builder.respirationMetrics(respirationModel.evaluate(time_ms));
        builder.visionMetrics(visionModel.evaluate(time_ms));
        return builder.build();
    }
    /**
     * Activate the Agent to perform a function at the specified time.
     *
     * @param currentTime_ms the simulation time.
     * @return a {@code ProcessingResult} containing internal and external {@code Message}s
     */
    @Override
    public ProcessingResult tick(long currentTime_ms) {
        PhysiologyState physiologyState = getState(currentTime_ms);
        return ProcessingResult.empty();
    }
    /**
     * Provides a {@code List} of all of the message content {@code Class}es to which this
     * {@code Handle} responds.
     * <p>
     * Handlers can respond to multiple message classes. Only one Handler may be associated with
     * a message class.
     *
     * @return a {@code List} of {@code Class} objects.
     */
    @Override
    public List<Class<? extends HandlerContent>> contentHandled() {
        return List.of();
    }
    /**
     * Performs the processing associated with the given argument at the specified time.
     * <p>
     * Both the scheduled time and current simulation time are reported. For an {@code EntityEven
t}
     * the scheduled time value will be less than or equal the current simulation time. For an
     * {@code EntityAction} scheduled time will be the same as the current time. Providing both t
imes
     * enables the Handler to adjust for any time difference between when an event was intended t
o occur
     * and when it was processed.
     *
     * @param arg              the message content
     * @param scheduledTime_ms the time for which an event was scheduled.
     * @param currentTime_ms   the current simulation time
     * @param from             the sources of the {@code Message}
     * @param id               the message sequence numner
     * @param respondTo        to where to send the results.
     * @return a {@code ProcessingResult} that contains the results of the processing
     */
    @Override
    public ProcessingResult process(HandlerContent arg, long scheduledTime_ms, long currentTime_m
s, Address from, long id, Address respondTo) {
        return buildUnsupportedMessageResponse(arg,id, getAddress(), new Throwable());
    }
}
```

A.84 SSTAF/src/features/simplePhysiologyAgent/mil.sstaf.physiology.agent/src/main/java/module-info.java

```
import mil.sstaf.core.features.Agent;
import mil.sstaf.core.features.Feature;
import mil.sstaf.core.features.Handler;
import mil.sstaf.physiology.agent.PhysiologyAgent;
import mil.sstaf.physiology.api.PhysiologyManagement;
module mil.sstaf.physiology.agent {
    exports mil.sstaf.physiology.agent;
    requires mil.sstaf.core;
    requires mil.devcom_sc.ansur.messages;
    requires mil.devcom_sc.ansur.api;
    requires mil.sstaf.physiology.api;
    requires mil.sstaf.physiology.models.api;
    requires org.slf4j;

    requires mil.sstaf.blackboard.api;
    provides PhysiologyManagement with PhysiologyAgent;
    provides Feature with PhysiologyAgent;
    provides Handler with PhysiologyAgent;
    provides Agent with PhysiologyAgent;
    opens mil.sstaf.physiology.agent to mil.sstaf.core;
}
```

A.85 SSTAF/src/features/simplePhysiologyAgent/mil.sstaf.physiology.agent/src/test/java/mil/sstaf/physiology/agent/PhysiologyAgentTest.java

```
package mil.sstaf.physiology.agent;
import mil.devcom_sc.ansur.messages.Handedness;
import mil.devcom_sc.ansur.messages.Sex;
import mil.devcom_sc.ansur.messages.ValueKey;
import mil.sstaf.core.entity.EntityHandle;
import mil.sstaf.core.features.Loaders;
import mil.sstaf.core.features.Resolver;
import mil.sstaf.core.util.Injector;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
public class PhysiologyAgentTest {
//    private PhysiologyAgent agent;
//
//    @BeforeEach
//    void setUp() {
//        try {
//            System.setProperty("sstaf.preloadFeatureClasses", "true");
//            Loaders.preloadFeatureClasses(
//                    "mil.sstaf.blackboard.inmem.InMemBlackboard",
//                    "mil.devcom_sc.ansur.handler.ANSURIIHandler",
//                    "mil.sstaf.physiology.models.cognition.CognitionModelImpl",
//                    "mil.sstaf.physiology.models.cardiovascular.CardioModelImpl",
//                    "mil.sstaf.physiology.models.respiration.RespirationModelImpl",
//                    "mil.sstaf.physiology.models.hydration.HydrationModelImpl",
//                    "mil.sstaf.physiology.models.musculature.MusculatureModelImpl",
//                    "mil.sstaf.physiology.models.energy.EnergyModelImpl",
//                    "mil.sstaf.physiology.models.vision.VisionModelImpl"
//            );
//        } catch (ClassNotFoundException e) {
//            e.printStackTrace();
//            fail();
//        }
//
//        agent = new PhysiologyAgent();
//        EntityHandle owner = EntityHandle.makeDummyHandle();
//        Injector.inject(agent, owner);
//        JSONObject config = new JSONObject();
//        JSONObject aconfig = new JSONObject();
//        aconfig.put(ValueKey.AGE.name(), 25);
//        aconfig.put(ValueKey.WRITING_PREFERENCE.name(), Handedness.RIGHT.name());
//        aconfig.put(ValueKey.STATURE.name(), 175.0);
//        aconfig.put(ValueKey.WEIGHT_KG.name(), 100.0);
//        aconfig.put(ValueKey.GENDER.name(), Sex.MALE.name());
//        aconfig.put(ValueKey.SPAN.name(), 175.00);
//        config.put("Simple Anthropometry", aconfig);
//        Resolver resolver = Resolver.makeTransientResolver(config);
//        resolver.resolveDependencies(agent);
//    }
//
//    @Test
//    void simpleTest() {
//        assertNotNull(agent);
//        assertDoesNotThrow(agent::init);
//    }
}
```

A.86 SSTAF/src/features/simplePhysiologyAgent/mil.sstaf.physiology.agent/src/test/resources/log4j2-test.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="INFO">
    <Appenders>
        <Console name="Console" target="SYSTEM_OUT">
            <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"/>
        </Console>
    </Appenders>
    <Loggers>
        <Logger name="mil.sstaf.core.features.Loaders" level="trace" additivity="false">
            <AppenderRef ref="Console"/>
        </Logger>
        <Logger name="mil.sstaf.core.features.Resolver" level="trace" additivity="false">
            <AppenderRef ref="Console"/>
        </Logger>
        <Logger name="mil.sstaf.core.features.FeatureLoader" level="trace" additivity="false">
            <AppenderRef ref="Console"/>
        </Logger>
        <Root level="info">
            <AppenderRef ref="Console"/>
        </Root>
    </Loggers>
</Configuration>
```

A.87 SSTAF/src/features/simplePhysiologyAgent/mil.sstaf.physiology.api/build.gradle

```
plugins {
    id 'java-library'
}
ext.moduleName = 'mil.sstaf.physiology.api'
dependencies {
    implementation group: 'org.slf4j', name: 'slf4j-api', version: '1.7.32'
    implementation project(':framework:mil.sstaf.core')
    implementation project(':features:support:mil.sstaf.blackboard.api')
    implementation project(':features:ansurHandler:mil.devcom_sc.ansur.messages')
    implementation project(':features:ansurHandler:mil.devcom_sc.ansur.api')
    implementation project(':features:simplePhysiologyAgent:mil.sstaf.physiology.models.api')
}
```

A.88 SSTAF/src/features/simplePhysiologyAgent/mil.sstaf.physiology.api/src/main/java/mil/sstaf/physiology/api/PhysiologyManagement.java

```
package mil.sstaf.physiology.api;
import mil.sstaf.core.features.Feature;
/**
 * Interface for accessing the aggregated state of the physiology sub-models.
 */
public interface PhysiologyManagement extends Feature {
    /**
     * Calculates and provides the state of all of the physiology sub-models at the provided time
.
     *
     * @param time_ms the simulation time at which to evaluate the models
     * @return a new PhysiologyState record.
     */
    PhysiologyState getState(long time_ms);
}
```

A.89 SSTAF/src/features/simplePhysiologyAgent/mil.sstaf.physiology.api/src/main/java/mil/sstaf/physiology/api/PhysiologyState.java

```
package mil.sstaf.physiology.api;
import lombok.Data;
import lombok.EqualsAndHashCode;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import mil.sstaf.core.state.LabeledState;
import mil.sstaf.core.state.StateProperty;
import mil.sstaf.physiology.models.api.*;
@Data
@EqualsAndHashCode(callSuper = true)
@SuperBuilder
@Jacksonized
public class PhysiologyState extends LabeledState {
    public final CognitionMetrics cognitionMetrics;
    public final CardiovascularMetrics cardiovascularMetrics;
    public final MusculatureMetrics musculatureMetrics;
    public final EnergyMetrics energyMetrics;
    public final HydrationMetrics hydrationMetrics;
    public final RespirationMetrics respirationMetrics;
    public final VisionMetrics visionMetrics;
    public final Boolean allowNullMembers;
    @StateProperty(headerLabel = "Cognition")
    public CognitionMetrics getCognitionMetrics() {
        return cognitionMetrics;
    }
    @StateProperty(headerLabel = "Cardiovascular")
    public CardiovascularMetrics getCardiovascularMetrics() {
        return cardiovascularMetrics;
    }
    @StateProperty(headerLabel = "Musculature")
    public MusculatureMetrics getMusculatureMetrics() {
        return musculatureMetrics;
    }
    @StateProperty(headerLabel = "Energy")
    public EnergyMetrics getEnergyMetrics() {
        return energyMetrics;
    }
    @StateProperty(headerLabel = "Hydration")
    public HydrationMetrics getHydrationMetrics() {
        return hydrationMetrics;
    }
    @StateProperty(headerLabel = "Respiration")
    public RespirationMetrics getRespirationMetrics() {
        return respirationMetrics;
    }
    @StateProperty(headerLabel = "Vision")
    public VisionMetrics getVisionMetrics() {
        return visionMetrics;
    }
}
```

A.90 SSTAF/src/features/simplePhysiologyAgent/mil.sstaf.physiology.api/src/main/java/module-info.java

```
module mil.sstaf.physiology.api {
    exports mil.sstaf.physiology.api;
    requires mil.sstaf.core;
    requires mil.sstaf.physiology.models.api;

    requires lombok;
}
```

A.91 SSTAF/src/features/simplePhysiologyAgent/mil.sstaf.physiology.models.api/build.gradle

```
ext.moduleName = 'mil.sstaf.physiology.models.api'
dependencies {
    implementation project(':framework:mil.sstaf.core')
    implementation project(':features:support:mil.sstaf.blackboard.api')
    implementation project(':features:ansurHandler:mil.devcom_sc.ansur.messages')
    implementation project(':features:ansurHandler:mil.devcom_sc.ansur.api')
    implementation project(':features:equipment:mil.devcom_dac.equipment.api')
}
```

A.92 SSTAF/src/features/simplePhysiologyAgent/mil.sstaf.physiology.models.api/src/main/java/mil/sstaf/physiology/models/api/CardiovascularMetrics.java

```
package mil.sstaf.physiology.models.api;
import mil.sstaf.core.state.LabeledState;
import mil.sstaf.core.state.State;
import mil.sstaf.core.state.StateProperty;
import lombok.Data;
import lombok.EqualsAndHashCode;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
@Data
@EqualsAndHashCode(callSuper = true)
@SuperBuilder
@Jacksonized
public class CardiovascularMetrics extends LabeledState {
    private final double pulseRatio;
    @StateProperty(headerLabel = "Pulse Ratio")
    public double getPulseRatio() {
        return pulseRatio;
    }
}
```

A.93 SSTAF/src/features/simplePhysiologyAgent/mil.sstaf.physiology.models.api/src/main/java/mil/sstaf/physiology/models/api/CardiovascularModel.java

```
package mil.sstaf.physiology.models.api;
public interface CardiovascularModel extends PhysiologyModel<CardiovascularMetrics> {
}
```

A.94 SSTAF/src/features/simplePhysiologyAgent/mil.sstaf.physiology.models.api/src/main/java/mil/sstaf/physiology/models/api/CognitionMetrics.java

```
package mil.sstaf.physiology.models.api;
import lombok.Data;
import lombok.EqualsAndHashCode;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import mil.sstaf.core.state.LabeledState;
import mil.sstaf.core.state.State;
import mil.sstaf.core.state.StateProperty;
@Data
@EqualsAndHashCode(callSuper = true)
@SuperBuilder
@Jacksonized
public class CognitionMetrics extends LabeledState {
    private final double smarts;
    @StateProperty(headerLabel = "Cognitive Ratio")
    public double getSmarts() {
        return smarts;
    }
}
```

A.95 SSTAF/src/features/simplePhysiologyAgent/mil.sstaf.physiology.models.api/src/main/java/mil/sstaf/physiology/models/api/CognitionModel.java

```
package mil.sstaf.physiology.models.api;
public interface CognitionModel extends PhysiologyModel<CognitionMetrics> {
}
```

A.96 SSTAF/src/features/simplePhysiologyAgent/mil.sstaf.physiology.models.api/src/main/java/mil/sstaf/physiology/models/api/EnergyMetrics.java

```
package mil.sstaf.physiology.models.api;
import lombok.Data;
import lombok.EqualsAndHashCode;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import mil.sstaf.core.state.LabeledState;
import mil.sstaf.core.state.StateProperty;
@Data
@EqualsAndHashCode(callSuper = true)
@SuperBuilder
@Jacksonized
public class EnergyMetrics extends LabeledState {
    private final double calorieReserve;
    @StateProperty(headerLabel = "Calorie Reserve")
    public double getCalorieReserve() {
        return calorieReserve;
    }
}
```

A.97 SSTAF/src/features/simplePhysiologyAgent/mil.sstaf.physiology.models.api/src/main/java/mil/sstaf/physiology/models/api/EnergyModel.java

```
package mil.sstaf.physiology.models.api;
public interface EnergyModel extends PhysiologyModel<EnergyMetrics> {
}
```

A.98 SSTAF/src/features/simplePhysiologyAgent/mil.sstaf.physiology.models.api/src/main/java/mil/sstaf/physiology/models/api/HydrationMetrics.java

```
package mil.sstaf.physiology.models.api;

import lombok.Data;
import lombok.EqualsAndHashCode;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import mil.sstaf.core.state.LabeledState;
import mil.sstaf.core.state.State;
import mil.sstaf.core.state.StateProperty;
@Data
@EqualsAndHashCode(callSuper = true)
@SuperBuilder
@Jacksonized
public class HydrationMetrics extends LabeledState {
    private final double hydrationLevel;
    @StateProperty(headerLabel = "Hydration Level")
    public double getHydrationLevel() {
        return hydrationLevel;
    }
}
```

A.99 SSTAF/src/features/simplePhysiologyAgent/mil.sstaf.physiology.models.api/src/main/java/mil/sstaf/physiology/models/api/HydrationModel.java

```
package mil.sstaf.physiology.models.api;
public interface HydrationModel extends PhysiologyModel<HydrationMetrics> {
}
```

A.100 SSTAF/src/features/simplePhysiologyAgent/mil.sstaf.physiology.models.api/src/main/java/mil/sstaf/physiology/models/api/MusculatureMetrics.java

```
package mil.sstaf.physiology.models.api;
import lombok.Data;
import lombok.EqualsAndHashCode;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import mil.sstaf.core.state.LabeledState;
import mil.sstaf.core.state.State;
import mil.sstaf.core.state.StateProperty;
import mil.devcom_sc.ansur.messages.Handedness;
@Data
@EqualsAndHashCode(callSuper = true)
@SuperBuilder
@Jacksonized
public class MusculatureMetrics extends LabeledState {
    private final double rightArmMaxStrength_N;
    private final double righArmMaxImpulse_Ns;
    private final double leftArmMaxStrength_N;
    private final double leftArmMaxImpulse_Ns;
    private final double rightArmStrength_N;
    private final boolean rightArmFatigued;
    private final double leftArmStrength_N;
    private final boolean leftArmFatigued;
    private final double rightArmLoad_N;
    private final double leftArmLoad_N;
    private final double rightRemainingImpulse_Ns;
    private final double leftRemainingImpulse_Ns;
    public final Handedness handedness;
    @StateProperty(headerLabel = "Right Arm Strength [N]")
    public double getRightArmStrength_N() {
        return rightArmStrength_N;
    }
    @StateProperty(headerLabel = "Right Arm Fatigued")
    public boolean isRightArmFatigued() {
        return rightArmFatigued;
    }
    @StateProperty(headerLabel = "Left Arm Strength [N]")
    public double getLeftArmStrength_N() {
        return leftArmStrength_N;
    }
    @StateProperty(headerLabel = "Left Arm Fatigued")
    public boolean isLeftArmFatigued() {
        return leftArmFatigued;
    }
    @StateProperty(headerLabel = "Non-dominant Arm Strength [N]")
    public double getNondominantArmStrength() {
        return handedness == Handedness.RIGHT ? leftArmStrength_N : rightArmStrength_N;
    }
    @StateProperty(headerLabel = "Dominant Arm Strength [N]")
    public double getDominantArmStrength() {
        return handedness == Handedness.RIGHT ? rightArmStrength_N : leftArmStrength_N;
    }
    @StateProperty(headerLabel = "Dominant Arm Max Strength [N]")
    public double getDominantArmMaxStrength() {
        return handedness == Handedness.RIGHT ? rightArmMaxStrength_N : leftArmMaxStrength_N;
    }
    @StateProperty(headerLabel = "Non-dominant Arm Max Strength [N]")
    public double getNondominantArmMaxStrength() {
        return handedness == Handedness.RIGHT ? leftArmMaxStrength_N : rightArmMaxStrength_N;
    }
    @StateProperty(headerLabel = "Right Arm Load [N]")
```

```
    public double getRightArmLoad_N() {
        return rightArmLoad_N;
    }
    @StateProperty(headerLabel = "Left Arm Load [N]")
    public double getLeftArmLoad_N() {
        return leftArmLoad_N;
    }
    @StateProperty(headerLabel = "Dominant Arm Load [N]")
    public double getDominantArmLoad() {
        return handedness == Handedness.RIGHT ? rightArmLoad_N : leftArmLoad_N;
    }
    @StateProperty(headerLabel = "Non-dominant Arm Load [N]")
    public double getNondominantArmLoad() {
        return handedness == Handedness.RIGHT ? leftArmLoad_N : rightArmLoad_N;
    }
    @StateProperty(headerLabel = "Right Arm Remaining Impulse [Ns]")
    public double getRightRemainingImpulse_Ns() {
        return rightRemainingImpulse_Ns;
    }
    @StateProperty(headerLabel = "Left Arm Remaining Impulse [Ns]")
    public double getLeftRemainingImpulse_Ns() {
        return leftRemainingImpulse_Ns;
    }
}
```

A.101 SSTAF/src/features/simplePhysiologyAgent/mil.sstaf.physiology.models.api/src/main/java/mil/sstaf/physiology/models/api/MusculatureModel.java

```
package mil.sstaf.physiology.models.api;
import mil.devcom_dac.equipment.api.Item;
/**
 * Muscle model
 */
public interface MusculatureModel extends PhysiologyModel<MusculatureMetrics> {
    /**
     * Sets a static load on each arm
     *
     * @param simTime_ms the time at which the load is applied
     * @param leffArm_N  the load on the left arm
     * @param rightArm_N i
     */
    void setArmLoads_N(long simTime_ms, double leffArm_N, double rightArm_N);
    /**
     * Configures the {@code MusculatureModel} to get the current arm loading from the weight of
     * an {@code Item}. This enables arm strength to be modified by a changeable load such as a g
un.
     *
     * @param simTime_ms             the simulation time at which the {@code Item} is added to th
e arms
     * @param item                   the {@code Item} that is being carried
     * @param dominantArmFraction    the fraction of the {@code Item} weight carried by the domin
ant arm
     * @param nondominantArmFraction the fraction of the {@code Item} weight carried by the non-d
ominant arm
     */
    void setArmItem(long simTime_ms, Item item, double dominantArmFraction, double nondominantArm
Fraction);
}
```

A.102 SSTAF/src/features/simplePhysiologyAgent/mil.sstaf.physiology.models.api/src/main/java/mil/sstaf/physiology/models/api/PhysiologyModel.java

```
package mil.sstaf.physiology.models.api;
import mil.sstaf.core.features.Feature;
import mil.sstaf.core.state.State;
public interface PhysiologyModel<T extends State> extends Feature {
    T evaluate(long time_ms);
}
```

A.103 SSTAF/src/features/simplePhysiologyAgent/mil.sstaf.physiology.models.api/src/main/java/mil/sstaf/physiology/models/api/RespirationMetrics.java

```
package mil.sstaf.physiology.models.api;
import lombok.Data;
import lombok.EqualsAndHashCode;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import mil.sstaf.core.state.LabeledState;
import mil.sstaf.core.state.State;
import mil.sstaf.core.state.StateProperty;
@Data
@EqualsAndHashCode(callSuper = true)
@SuperBuilder
@Jacksonized
public class RespirationMetrics extends LabeledState {
    private final double spO2;
    @StateProperty(headerLabel = "spO2")
    public double getSpO2() {
        return spO2;
    }
}
```

A.104 SSTAF/src/features/simplePhysiologyAgent/mil.sstaf.physiology.models.api/src/main/java/mil/sstaf/physiology/models/api/RespirationModel.java

```
package mil.sstaf.physiology.models.api;
public interface RespirationModel extends PhysiologyModel<RespirationMetrics> {
}
```

A.105 SSTAF/src/features/simplePhysiologyAgent/mil.sstaf.physiology.models.api/src/main/java/mil/sstaf/physiology/models/api/VisionMetrics.java

```
package mil.sstaf.physiology.models.api;
import lombok.Data;
import lombok.EqualsAndHashCode;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import mil.sstaf.core.state.LabeledState;
import mil.sstaf.core.state.State;
import mil.sstaf.core.state.StateProperty;
/**
 * Defines the current state of vision
 */
@Data
@EqualsAndHashCode(callSuper = true)
@SuperBuilder
@Jacksonized
public class VisionMetrics extends LabeledState {
    private final double angularResolution_mils;
    private final double visionRatio;
    @StateProperty(headerLabel = "Angular Resolution (mils)")
    public double getAngularResolution_mils() {
        return angularResolution_mils;
    }
    @StateProperty(headerLabel = "Vision Ratio")
    public double getVisionRatio() {
        return visionRatio;
    }
}
```

A.106 SSTAF/src/features/simplePhysiologyAgent/mil.sstaf.physiology.models.api/src/main/java/mil/sstaf/physiology/models/api/VisionModel.java

```
package mil.sstaf.physiology.models.api;
public interface VisionModel extends PhysiologyModel<VisionMetrics> {
}
```

A.107 SSTAF/src/features/simplePhysiologyAgent/mil.sstaf.physiology.models.api/src/main/java/module-info.java

```
module mil.sstaf.physiology.models.api {
    exports mil.sstaf.physiology.models.api;
    requires mil.sstaf.core;
    requires mil.devcom_sc.ansur.api;
    requires mil.devcom_dac.equipment.api;
}
```

A.108 SSTAF/src/features/simplePhysiologyAgent/mil.sstaf.physiology.models.cardiovascular/build.gradle

```
plugins {
    id 'java-library'
}
ext.moduleName = 'mil.sstaf.physiology.models.cardiovascular'
dependencies {
    implementation project(':framework:mil.sstaf.core')
    implementation project(':features:support:mil.sstaf.blackboard.api')
    implementation project(':features:ansurHandler:mil.devcom_sc.ansur.messages')
    implementation project(':features:ansurHandler:mil.devcom_sc.ansur.api')
    implementation project(':features:simplePhysiologyAgent:mil.sstaf.physiology.api')
    implementation project(':features:simplePhysiologyAgent:mil.sstaf.physiology.models.api')
}
```

A.109 SSTAF/src/features/simplePhysiologyAgent/mil.sstaf.physiology.models.cardiovascular/src/main/java/mil/sstaf/physiology/models/cardiovascular/CardioModelImpl.java

```
package mil.sstaf.physiology.models.cardiovascular;
import mil.sstaf.blackboard.api.Blackboard;
import mil.sstaf.core.features.BaseFeature;
import mil.sstaf.core.features.FeatureConfiguration;
import mil.sstaf.core.features.Requires;
import mil.sstaf.core.util.SSTAFException;
import mil.sstaf.physiology.models.api.CardiovascularMetrics;
import mil.sstaf.physiology.models.api.CardiovascularModel;
import org.apache.commons.math3.random.MersenneTwister;
public class CardioModelImpl extends BaseFeature implements CardiovascularModel {
    public static final String FEATURE_NAME = "Cardiovascular Model";
    public static final int MAJOR_VERSION = 0;
    public static final int MINOR_VERSION = 1;
    public static final int PATCH_VERSION = 0;
    public static final String BK_CARDIOVASCULAR_METRICS = "Cardiovascular Metrics";
    private final MersenneTwister rng = new MersenneTwister();
    @Requires
    Blackboard blackboard;
    public CardioModelImpl() {
        super(FEATURE_NAME, MAJOR_VERSION, MINOR_VERSION, PATCH_VERSION,
                false, "Heart!");
    }
    @Override
    public CardiovascularMetrics evaluate(long time_ms) {
        double pr = 1.0 + Math.abs(rng.nextGaussian());
        var cm = CardiovascularMetrics.builder().pulseRatio(pr).timestamp_ms(time_ms).build();
        blackboard.addEntry(BK_CARDIOVASCULAR_METRICS, cm, time_ms);
        return cm;
    }
    /**
     * Initializes this {@code Provider}
     *
     * @throws SSTAFException if an error occurs.
     */
    @Override
    public void init() throws SSTAFException {
        super.init();
    }
    /**
     * Sets the configuration for this provider.
     * <p>
     * The configuration is applied when {@code init()} is invoked.
     *
     * @param configuration
     */
    @Override
    public void configure(FeatureConfiguration configuration) {
        super.configure(configuration);
        rng.setSeed(configuration.getSeed());
    }
}
```

A.110 SSTAF/src/features/simplePhysiologyAgent/mil.sstaf.physiology.models.cardiovascular/src/main/java/module-info.java

```
import mil.sstaf.core.features.Feature;
import mil.sstaf.physiology.models.api.CardiovascularModel;
import mil.sstaf.physiology.models.cardiovascular.CardioModelImpl;
module mil.sstaf.physiology.models.cardiovascular {
    exports mil.sstaf.physiology.models.cardiovascular;
    requires mil.sstaf.core;
    requires mil.sstaf.physiology.models.api;

    requires org.slf4j;
    requires mil.sstaf.blackboard.api;
    provides CardiovascularModel with CardioModelImpl;
    provides Feature with CardioModelImpl;
    opens mil.sstaf.physiology.models.cardiovascular to mil.sstaf.core;
}
```

A.111 SSTAF/src/features/simplePhysiologyAgent/mil.sstaf.physiology.models.cognition/build.gradle

```
plugins {
    id 'java-library'
}
ext.moduleName = 'mil.sstaf.physiology.models.cognition'
dependencies {
    implementation project(':framework:mil.sstaf.core')
    implementation project(':features:support:mil.sstaf.blackboard.api')
    implementation project(':features:ansurHandler:mil.devcom_sc.ansur.messages')
    implementation project(':features:ansurHandler:mil.devcom_sc.ansur.api')
    implementation project(':features:simplePhysiologyAgent:mil.sstaf.physiology.api')
    implementation project(':features:simplePhysiologyAgent:mil.sstaf.physiology.models.api')
}
```

A.112 SSTAF/src/features/simplePhysiologyAgent/mil.sstaf.physiology.models.cognition/src/main/java/mil/sstaf/physiology/models/cognition/CognitionModelImpl.java

```
package mil.sstaf.physiology.models.cognition;
import mil.sstaf.blackboard.api.Blackboard;
import mil.sstaf.core.features.BaseFeature;
import mil.sstaf.core.features.FeatureConfiguration;
import mil.sstaf.core.features.Requires;
import mil.sstaf.core.util.SSTAFException;
import mil.sstaf.physiology.models.api.CognitionMetrics;
import mil.sstaf.physiology.models.api.CognitionModel;
public class CognitionModelImpl extends BaseFeature implements CognitionModel {
    public static final String FEATURE_NAME = "Cognition Model";
    public static final String BK_COGNITION_METRICS = "Cognition Metrics";
    public static final int MAJOR_VERSION = 0;
    public static final int MINOR_VERSION = 1;
    public static final int PATCH_VERSION = 0;
    @Requires
    Blackboard blackboard;
    public CognitionModelImpl() {
        super(FEATURE_NAME, MAJOR_VERSION, MINOR_VERSION, PATCH_VERSION,
                false, "Brains!");
    }
    @Override
    public CognitionMetrics evaluate(long time_ms) {
        long maxAwake = 36 * 3600 * 1000;
        double smarts = (time_ms <= maxAwake) ? 1.0 - (double) time_ms / maxAwake : 0.0;
        var cm = CognitionMetrics.builder().smarts(smarts).build();
        blackboard.addEntry(BK_COGNITION_METRICS, cm, time_ms);
        return cm;
    }
    /**
     * Initializes this {@code Provider}
     *
     * @throws SSTAFException if an error occurs.
     */
    @Override
    public void init() throws SSTAFException {
        super.init();
    }
    /**
     * Sets the configuration for this provider.
     * <p>
     * The configuration is applied when {@code init()} is invoked.
     * @param configuration
     *
     */
    @Override
    public void configure(FeatureConfiguration configuration) {
        super.configure(configuration);
    }
}
```

A.113 SSTAF/src/features/simplePhysiologyAgent/mil.sstaf.physiology.models.cognition/src/main/java/module-info.java

```
import mil.sstaf.core.features.Feature;
import mil.sstaf.physiology.models.api.CognitionModel;
import mil.sstaf.physiology.models.cognition.CognitionModelImpl;

module mil.sstaf.physiology.models.cognition {
    exports mil.sstaf.physiology.models.cognition;
    requires mil.sstaf.core;
    requires mil.sstaf.blackboard.api;
    requires mil.sstaf.physiology.models.api;

    requires org.slf4j;
    provides CognitionModel with CognitionModelImpl;
    provides Feature with CognitionModelImpl;
    opens mil.sstaf.physiology.models.cognition to mil.sstaf.core;
}
```

A.114 SSTAF/src/features/simplePhysiologyAgent/mil.sstaf.physiology.models.energy/build.gradle

```
plugins {
    id 'java-library'
}
ext.moduleName = 'mil.sstaf.physiology.models.energy'
dependencies {
    implementation project(':framework:mil.sstaf.core')
    implementation project(':features:support:mil.sstaf.blackboard.api')
    implementation project(':features:ansurHandler:mil.devcom_sc.ansur.messages')
    implementation project(':features:ansurHandler:mil.devcom_sc.ansur.api')
    implementation project(':features:simplePhysiologyAgent:mil.sstaf.physiology.api')
    implementation project(':features:simplePhysiologyAgent:mil.sstaf.physiology.models.api')
}
```

A.115 SSTAF/src/features/simplePhysiologyAgent/mil.sstaf.physiology.models.energy/src/main/java/mil/sstaf/physiology/models/energy/EnergyModelImpl.java

```
package mil.sstaf.physiology.models.energy;
import mil.sstaf.blackboard.api.Blackboard;
import mil.sstaf.core.features.BaseFeature;
import mil.sstaf.core.features.FeatureConfiguration;
import mil.sstaf.core.features.Requires;
import mil.sstaf.core.util.SSTAFException;
import mil.sstaf.physiology.models.api.EnergyMetrics;
import mil.sstaf.physiology.models.api.EnergyModel;
import org.apache.commons.math3.random.MersenneTwister;
public class EnergyModelImpl extends BaseFeature implements EnergyModel {
    public static final String FEATURE_NAME = "Energy Model";
    public static final String BK_ENERGY_METRICS = "Energy Metrics";
    public static final int MAJOR_VERSION = 0;
    public static final int MINOR_VERSION = 1;
    public static final int PATCH_VERSION = 0;
    @Requires
    Blackboard blackboard;
    private final MersenneTwister rng = new MersenneTwister();
    public EnergyModelImpl() {
        super(FEATURE_NAME, MAJOR_VERSION, MINOR_VERSION, PATCH_VERSION,
                false, "Energy! Metabolism! Calories!");
    }
    @Override
    public EnergyMetrics evaluate(long time_ms) {
        var em = EnergyMetrics.builder()
                .calorieReserve(rng.nextDouble() * 10000.0)
                .timestamp_ms(time_ms)
                .build();
        blackboard.addEntry(BK_ENERGY_METRICS, em, time_ms);
        return em;
    }
    /**
     * Initializes this {@code Provider}
     *
     * @throws SSTAFException if an error occurs.
     */
    @Override
    public void init() throws SSTAFException {
        super.init();
    }
    /**
     * Sets the configuration for this provider.
     * <p>
     * The configuration is applied when {@code init()} is invoked.
     * @param configuration
     *
     */
    @Override
    public void configure(FeatureConfiguration configuration) {
        super.configure(configuration);
        rng.setSeed(configuration.getSeed());
    }
}
```

A.116 SSTAF/src/features/simplePhysiologyAgent/mil.sstaf.physiology.models.energy/src/main/java/module-info.java

```
import mil.sstaf.core.features.Feature;
import mil.sstaf.physiology.models.api.EnergyModel;
import mil.sstaf.physiology.models.energy.EnergyModelImpl;
module mil.sstaf.physiology.models.energy {
    exports mil.sstaf.physiology.models.energy;
    requires mil.sstaf.core;
    requires mil.sstaf.physiology.models.api;

    requires org.slf4j;
    requires mil.sstaf.blackboard.api;
    provides EnergyModel with EnergyModelImpl;
    provides Feature with EnergyModelImpl;
    opens mil.sstaf.physiology.models.energy to mil.sstaf.core;
}
```

A.117 SSTAF/src/features/simplePhysiologyAgent/mil.sstaf.physiology.models.hydration/build.gradle

```
plugins {
    id 'java-library'
}
ext.moduleName = 'mil.sstaf.physiology.models.hydration'
dependencies {
    implementation project(':framework:mil.sstaf.core')
    implementation project(':features:support:mil.sstaf.blackboard.api')
    implementation project(':features:ansurHandler:mil.devcom_sc.ansur.messages')
    implementation project(':features:ansurHandler:mil.devcom_sc.ansur.api')
    implementation project(':features:simplePhysiologyAgent:mil.sstaf.physiology.api')
    implementation project(':features:simplePhysiologyAgent:mil.sstaf.physiology.models.api')
}
```

A.118 SSTAF/src/features/simplePhysiologyAgent/mil.sstaf.physiology.models.hydration/src/main/java/mil/sstaf/physiology/models/hydration/HydrationModelImpl.java

```
package mil.sstaf.physiology.models.hydration;
import mil.sstaf.blackboard.api.Blackboard;
import mil.sstaf.core.features.BaseFeature;
import mil.sstaf.core.features.FeatureConfiguration;
import mil.sstaf.core.features.Requires;
import mil.sstaf.core.util.SSTAFException;
import mil.sstaf.physiology.models.api.HydrationMetrics;
import mil.sstaf.physiology.models.api.HydrationModel;
public class HydrationModelImpl extends BaseFeature implements HydrationModel {
    public static final String FEATURE_NAME = "Hydration Model";
    public static final String BK_HYDRATION_METRICS = "Hydration Metrics";
    public static final int MAJOR_VERSION = 0;
    public static final int MINOR_VERSION = 1;
    public static final int PATCH_VERSION = 0;
    @Requires
    Blackboard blackboard;
    public HydrationModelImpl() {
        super(FEATURE_NAME, MAJOR_VERSION, MINOR_VERSION, PATCH_VERSION,
                false, "A model for hydration");
    }
    @Override
    public HydrationMetrics evaluate(long time_ms) {
        long maxTime = 72 * 3600 * 1000;
        double hyd = (time_ms <= maxTime) ? 1.0 - (double) time_ms / maxTime : 0.0;
        var hm = HydrationMetrics.builder().hydrationLevel(hyd).timestamp_ms(time_ms).build();
        blackboard.addEntry(BK_HYDRATION_METRICS, hm, time_ms);
        return hm;
    }
    /**
     * Initializes this {@code Provider}
     *
     * @throws SSTAFException if an error occurs.
     */
    @Override
    public void init() throws SSTAFException {
    }
    /**
     * Sets the configuration for this provider.
     * <p>
     * The configuration is applied when {@code init()} is invoked.
     * @param configuration
     *
     */
    @Override
    public void configure(FeatureConfiguration configuration) {
    }
}
```

A.119 SSTAF/src/features/simplePhysiologyAgent/mil.sstaf.physiology.models.hydration/src/main/java/module-info.java

```
import mil.sstaf.core.features.Feature;
import mil.sstaf.physiology.models.api.HydrationModel;
import mil.sstaf.physiology.models.hydration.HydrationModelImpl;
module mil.sstaf.physiology.models.hydration {
    exports mil.sstaf.physiology.models.hydration;
    requires mil.sstaf.core;
    requires mil.sstaf.physiology.models.api;

    requires org.slf4j;
    requires mil.sstaf.blackboard.api;
    provides HydrationModel with HydrationModelImpl;
    provides Feature with HydrationModelImpl;
    opens mil.sstaf.physiology.models.hydration to mil.sstaf.core;
}
```

A.120 SSTAF/src/features/simplePhysiologyAgent/mil.sstaf.physiology.models.musculature/build.gradle

```
ext.moduleName = 'mil.sstaf.physiology.models.musculature'
dependencies {
    implementation project(':framework:mil.sstaf.core')
    implementation project(':features:support:mil.sstaf.blackboard.api')
    implementation project(':features:ansurHandler:mil.devcom_sc.ansur.messages')
    implementation project(':features:ansurHandler:mil.devcom_sc.ansur.api')
    implementation project(':features:equipment:mil.devcom_dac.equipment.api')
    implementation project(':features:simplePhysiologyAgent::mil.sstaf.physiology.api')
    implementation project(':features:simplePhysiologyAgent:mil.sstaf.physiology.models.api')
}
testing {
    suites {
        integrationTest {
            dependencies {
                implementation project(':framework:mil.sstaf.core')
                implementation project(':features:support:mil.sstaf.blackboard.api')
                implementation project(':features:ansurHandler:mil.devcom_sc.ansur.messages')
                implementation project(':features:ansurHandler:mil.devcom_sc.ansur.api')
                implementation project(':features:simplePhysiologyAgent:mil.sstaf.physiology.api'
)
                implementation project(':features:simplePhysiologyAgent:mil.sstaf.physiology.mode
ls.api')
                runtimeOnly project(':features:support:mil.sstaf.blackboard.inmem')
                runtimeOnly project(':features:ansurHandler:mil.devcom_sc.ansur.handler')
            }
        }
    }
}
```

A.121 SSTAF/src/features/simplePhysiologyAgent/mil.sstaf.physiology.models.musculature/src/integrationTest/java/mil/sstaf/physiology/models/musculature/test/MusculatureModelImplTest.java

```
package mil.sstaf.physiology.models.musculature.test;
import mil.sstaf.core.entity.Address;
import mil.sstaf.core.entity.EntityHandle;
import mil.sstaf.core.features.Loaders;
import mil.sstaf.core.features.Resolver;
import mil.sstaf.core.util.Injector;
import mil.sstaf.core.util.PhysicalConstants;
import mil.sstaf.core.configuration.SSTAFConfiguration;
import mil.sstaf.physiology.models.api.MusculatureMetrics;
import mil.sstaf.physiology.models.api.MusculatureModel;
import mil.sstaf.physiology.models.musculature.MusculatureModelImpl;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import java.io.File;
import static org.junit.jupiter.api.Assertions.*;
public class MusculatureModelImplTest {
//    private static final Logger logger = LoggerFactory.getLogger(MusculatureModelImplTest.class
);
//
//    MusculatureModel muscleModel;
//    EntityHandle owner;
//    Address address;
//
//    @BeforeEach
//    public void setup() {
//        System.setProperty(SSTAFConfiguration.SSTAF_CONFIGURATION_PROPERTY,
//                "src" + File.separator +
//                        "integrationTest" + File.separator +
//                        "resources" + File.separator +
//                        "EmptyConfiguration.json");
//    }
//
//    void makeProvider(String configurationPath) {
//        muscleModel = Loaders.loadAsRef(MusculatureModel.class, "Muscle Model", 0, 1, false, Mo
duleLayer.boot());
//        JSONObject configuration =
//                JSONUtilities.loadObjectFromFile(configurationPath, null);
//        Resolver.makeTransientResolver(configuration).resolveDependencies(muscleModel);
//        owner = EntityHandle.makeDummyHandle();
//        Injector.inject(muscleModel, owner);
//        muscleModel.init();
//        address = Address.makeExternalAddress(owner);
//    }
//
//    @Test
//    void setupWorks() {
//        makeProvider("src/integrationTest/resources/testConfig1.json");
//        Assertions.assertNotNull(muscleModel);
//        MusculatureMetrics metrics = muscleModel.evaluate(0);
//        assertNotNull(metrics);
//        //
//        // Configuration
//        //
//        double d = 781.05063925;
//        double s = 661.5958356;
//        assertEquals(d, metrics.getDominantArmStrength());
```

```
//        assertEquals(s, metrics.getNondominantArmStrength());
//    }
//
//    @Test
//    void spanAdjustmentWorks() {
//        makeProvider("src/integrationTest/resources/testConfig2.json");
//        Assertions.assertNotNull(muscleModel);
//        MusculatureMetrics metrics = muscleModel.evaluate(0);
//        assertNotNull(metrics);
//        //
//        // Configuration
//        //
//
//        double d = 555.9480904805389;
//        double s = 470.9207354658683;
//        assertEquals(d, metrics.getDominantArmStrength(), 0.000001);  // Float fuzz
//        assertEquals(s, metrics.getNondominantArmStrength(), 0.000001);
//    }
//
//    @Test
//    void fatigueAndRestorationWork() {
//        makeProvider("src/integrationTest/resources/testConfig1.json");
//        double dMax = MusculatureModelImpl.DOMINANT_STRENGTH_FACTOR * 100 * PhysicalConstants.G
RAVITY;
//        double sMax = MusculatureModelImpl.NON_DOMINANT_STRENGTH_FACTOR * 100 * PhysicalConstan
ts.GRAVITY;
//        double d = dMax;
//        double s = sMax;
//        muscleModel.setArmLoads_N(0, 0.1 * s, 0.1 * d);
//
//        MusculatureMetrics metrics;
//        long time_ms = 0;
//        int fatigueLoops = 0;
//        do {
//            metrics = muscleModel.evaluate(time_ms);
//            d = metrics.getRightArmStrength_N();
//            s = metrics.getLeftArmStrength_N();
//            ++fatigueLoops;
//            time_ms += 10000.0;
//            if (logger.isTraceEnabled()) {
//                logger.trace("{}: right {}  {} left {}  {}", time_ms, d, metrics.isRightArmFati
gued(),
//                        s, metrics.isLeftArmFatigued());
//            }
//        } while (d > 0 || s > 0);
//
//        muscleModel.setArmLoads_N(time_ms, 0.0, 0.0);
//        int restoreLoops = 0;
//        do {
//            metrics = muscleModel.evaluate(time_ms);
//            d = metrics.getRightArmStrength_N();
//            s = metrics.getLeftArmStrength_N();
//            ++restoreLoops;
//            time_ms += 10000;
//            if (logger.isTraceEnabled()) {
//                logger.debug("{}: right {}  {} left {}  {}", time_ms, d, metrics.isRightArmFati
gued(),
//                        s, metrics.isLeftArmFatigued());
//            }
//        } while (d < dMax && s < sMax);
//
//        assertTrue(restoreLoops > fatigueLoops);
//    }
//
}
```

A.122 SSTAF/src/features/simplePhysiologyAgent/mil.sstaf.physiology.models.musculature/src/integrationTest/java/module-info.java

```
open module mil.sstaf.physiology.models.musculature.integrationTest {
    requires mil.sstaf.core;

    requires org.slf4j;
    requires org.junit.jupiter.api;
    requires mil.sstaf.physiology.models.api;
    requires mil.sstaf.physiology.models.musculature;
}
```

A.123 SSTAF/src/features/simplePhysiologyAgent/mil.sstaf.physiology.models.musculature/src/integrationTest/resources/EmptyConfiguration.json

```
{
  "modules": [
  ],
  "modulePaths": [
  ]
}
```

A.124 SSTAF/src/features/simplePhysiologyAgent/mil.sstaf.physiology.models.musculature/src/integrationTest/resources/log4j2-test.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="INFO">
    <Appenders>
        <Console name="Console" target="SYSTEM_OUT">
            <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"/>
        </Console>
    </Appenders>
    <Loggers>
        <Root level="debug">
            <AppenderRef ref="Console"/>
        </Root>
    </Loggers>
</Configuration>
```

A.125 SSTAF/src/features/simplePhysiologyAgent/mil.sstaf.physiology.models.musculature/src/integrationTest/resources/testConfig1.json

```
{
  "ANSUR Anthropometry": {
    "subjectID" : 11427
  }
}
```

A.126 SSTAF/src/features/simplePhysiologyAgent/mil.sstaf.physiology.models.musculature/src/integrationTest/resources/testConfig2.json

```
{
  "ANSUR Anthropometry": {
    "subjectID" : 13151
  }
}
```

A.127 SSTAF/src/features/simplePhysiologyAgent/mil.sstaf.physiology.models.musculature/src/main/java/mil/sstaf/physiology/models/musculature/MusculatureModelImpl.java

```
package mil.sstaf.physiology.models.musculature;
import mil.sstaf.blackboard.api.Blackboard;
import mil.sstaf.core.features.BaseFeature;
import mil.sstaf.core.features.FeatureConfiguration;
import mil.sstaf.core.features.Requires;
import mil.sstaf.core.util.PhysicalConstants;
import mil.sstaf.core.util.SSTAFException;
import mil.devcom_sc.ansur.api.ANSURIIAnthropometry;
import mil.devcom_sc.ansur.messages.Handedness;
import mil.devcom_dac.equipment.api.Item;
import mil.sstaf.physiology.models.api.MusculatureMetrics;
import mil.sstaf.physiology.models.api.MusculatureModel;
import org.slf4j.LoggerFactory;
import org.slf4j.Logger;
import java.util.Objects;
public class MusculatureModelImpl extends BaseFeature implements MusculatureModel {
    public static final String BK_MUSCLE_METRICS = "Muscle Metrics";
    public static final String FEATURE_NAME = "Muscle Model";
    public static final int MAJOR_VERSION = 0;
    public static final int MINOR_VERSION = 1;
    public static final int PATCH_VERSION = 0;
    public static final double DOMINANT_STRENGTH_FACTOR = 0.85;
    public static final double NON_DOMINANT_STRENGTH_FACTOR = 0.72;
    private static final Logger logger = LoggerFactory.getLogger(MusculatureModelImpl.class);
    @Requires
    Blackboard blackboard;
    @Requires()
    ANSURIIAnthropometry anthropometry;
    ArmModel rightArm = new ArmModel();
    ArmModel leftArm = new ArmModel();
    ArmModel dominantArm;
    ArmModel nonDominantArm;
    public MusculatureModelImpl() {
        super(FEATURE_NAME, MAJOR_VERSION, MINOR_VERSION, PATCH_VERSION, false, "Arms! Leg! Glute
s!");
    }
    /**
     * Initializes this {@code Provider}
     *
     * @throws SSTAFException if an error occurs.
     */
    @Override
    public void init() throws SSTAFException {
        super.init();
        Objects.requireNonNull(anthropometry, "Anthropometry is null");
        Objects.requireNonNull(blackboard, "Blackboard is null");
        if (anthropometry.getHandedness() == Handedness.RIGHT) {
            dominantArm = rightArm;
            nonDominantArm = leftArm;
        } else {
            dominantArm = leftArm;
            nonDominantArm = rightArm;
        }
        dominantArm.init(true);
        nonDominantArm.init(false);
    }
    /**
     * Configures the {@code MusculatureModel} to get the current arm loading from the weight of
     * an {@code Item}. This enables arm strength to be modified by a changeable load such as a g
un.
```

```
     *
     * @param simTime_ms             the simulation time at which the {@code Item} is added to th
e arms
     * @param item                   the {@code Item} that is being carried
     * @param dominantArmFraction    the fraction of the {@code Item} weight carried by the domin
ant arm
     * @param nonDominantArmFraction the fraction of the {@code Item} weight carried by the non-d
ominant arm
     */
    @Override
    public void setArmItem(long simTime_ms, Item item, double dominantArmFraction, double nonDomi
nantArmFraction) {
        if (logger.isDebugEnabled()) {
            logger.debug("{}:{} - setting arms item as {}(d={}/nd={})",
                    ownerHandle.getPath(), featureName, item.getName(), dominantArmFraction, nonD
ominantArmFraction);
        }
        dominantArm.setItem(simTime_ms, item, dominantArmFraction);
        nonDominantArm.setItem(simTime_ms, item, nonDominantArmFraction);
    }
    @Override
    public void setArmLoads_N(long simTime_ms, double leftArm_N, double rightArm_N) {
        calculateArmStrength(simTime_ms); // update current strength;
        leftArm.setLoad(simTime_ms, leftArm_N);
        rightArm.setLoad(simTime_ms, rightArm_N);
    }
    /**
     * Sets the configuration for this provider.
     * <p>
     * The configuration is applied when {@code init()} is invoked.
     * @param configuration
     *
     */
    @Override
    public void configure(FeatureConfiguration configuration) {
        super.configure(configuration);
    }
    void calculateArmStrength(long time_ms) {
        rightArm.calculateCurrentStrength(time_ms);
        leftArm.calculateCurrentStrength(time_ms);
    }
    @Override
    public MusculatureMetrics evaluate(long time_ms) {
        calculateArmStrength(time_ms);
        var builder = MusculatureMetrics.builder();
        builder.handedness(anthropometry.getHandedness());
        builder.rightArmMaxStrength_N(rightArm.getMaximumStrength_N());
        builder.righArmMaxImpulse_Ns(rightArm.getMaxImpulse_Ns());
        builder.rightArmLoad_N(rightArm.getLoad());
        builder.leftArmMaxStrength_N(leftArm.getMaximumStrength_N());
        builder.leftArmMaxImpulse_Ns(leftArm.getMaxImpulse_Ns());
        builder.leftArmLoad_N(leftArm.getLoad());
        builder.leftArmStrength_N(leftArm.getCurrentStrength_N(time_ms));
        builder.leftArmFatigued(leftArm.isFatigued(time_ms));
        builder.leftRemainingImpulse_Ns(leftArm.getRemainingImpulse_Ns(time_ms));
        builder.rightArmStrength_N(rightArm.getCurrentStrength_N(time_ms));
        builder.rightArmFatigued(rightArm.isFatigued(time_ms));
        builder.rightRemainingImpulse_Ns(rightArm.getRemainingImpulse_Ns(time_ms));
        MusculatureMetrics metrics = builder.build();
        blackboard.addEntry(BK_MUSCLE_METRICS, metrics, time_ms, time_ms + 20 * 60 * 1000);
        return metrics;
    }
    class ArmModel {
        public static final int STRENGTH_SCALE = 100;
        private double maximumStrength_N;
        private double maxImpulse_Ns;
        private double remainingImpulse_Ns;
        private double currentStrength_N;
```

```
        private boolean fatigued = false;
        private long updateTime_ms;
        private double baseLoad_N;
        private Item item = null;
        private double weightFraction = 0.0;
        void init(boolean isDominant) {
            //
            // Fake strength until later.
            //
            double weight = anthropometry.getWeight_kg() * PhysicalConstants.GRAVITY;
            double chestAdjustment = 1.0 + 4 * (anthropometry.getSpan_cm() / anthropometry.getHei
ght_cm() - 1.0);
            double dominantAdjustment = isDominant ? DOMINANT_STRENGTH_FACTOR : NON_DOMINANT_STRE
NGTH_FACTOR;
            double max = dominantAdjustment * chestAdjustment * weight;
            maximumStrength_N = max;
            currentStrength_N = max;
            maxImpulse_Ns = maximumStrength_N * STRENGTH_SCALE;
            remainingImpulse_Ns = maxImpulse_Ns;
            baseLoad_N = 0.0;
            updateTime_ms = 0;
        }
        void setItem(long time_ms, Item item, double weightFraction) {
            calculateCurrentStrength(time_ms);
            this.updateTime_ms = time_ms;
            this.item = item;
            this.weightFraction = weightFraction;
        }
        void setLoad(long time_ms, double load_N) {
            calculateCurrentStrength(time_ms);
            this.updateTime_ms = time_ms;
            this.baseLoad_N = load_N;
        }
        double getLoad() {
            double itemLoad = item == null ? 0.0 :
                    item.getMass_kg() * PhysicalConstants.GRAVITY * weightFraction;
            return itemLoad + baseLoad_N;
        }
        void calculateCurrentStrength(long time_ms) {
            if (time_ms > updateTime_ms) {
                long deltaT_ms = time_ms - updateTime_ms;
                double deltaT_s = deltaT_ms / 1000.0;
                double currentLoad_N = getLoad();
                if (currentLoad_N > 0) {
                    remainingImpulse_Ns = remainingImpulse_Ns - deltaT_s * currentLoad_N;
                    if (remainingImpulse_Ns <= 0) remainingImpulse_Ns = 0;
                } else {
                    remainingImpulse_Ns += deltaT_s * maximumStrength_N / (1.2 * STRENGTH_SCALE);
                    if (remainingImpulse_Ns >= maxImpulse_Ns) remainingImpulse_Ns = maxImpulse_Ns
;
                }
                currentStrength_N = remainingImpulse_Ns / STRENGTH_SCALE;
                fatigued = currentStrength_N <= currentLoad_N;
                updateTime_ms = time_ms;
            }
        }
        public double getMaximumStrength_N() {
            return maximumStrength_N;
        }
        public double getMaxImpulse_Ns() {
            return maxImpulse_Ns;
        }
        public double getRemainingImpulse_Ns(long time_ms) {
            calculateCurrentStrength(time_ms);
            return remainingImpulse_Ns;
        }
        public double getCurrentStrength_N(long time_ms) {
            calculateCurrentStrength(time_ms);
```

```
            return currentStrength_N;
        }
        public boolean isFatigued(long time_ms) {
            calculateCurrentStrength(time_ms);
            return fatigued;
        }
        public long getUpdateTime_ms() {
            return updateTime_ms;
        }
        public double getBaseLoad_N() {
            return baseLoad_N;
        }
        public Item getItem() {
            return item;
        }
        public double getWeightFraction() {
            return weightFraction;
        }
    }
}
```

A.128 SSTAF/src/features/simplePhysiologyAgent/mil.sstaf.physiology.models.musculature/src/main/java/module-info.java

```
import mil.sstaf.core.features.Feature;
import mil.sstaf.physiology.models.api.MusculatureModel;
import mil.sstaf.physiology.models.musculature.MusculatureModelImpl;
module mil.sstaf.physiology.models.musculature {
    exports mil.sstaf.physiology.models.musculature;
    requires mil.sstaf.core;
    requires mil.devcom_sc.ansur.messages;
    requires mil.devcom_sc.ansur.api;
    requires mil.devcom_dac.equipment.api;
    requires mil.sstaf.blackboard.api;
    requires mil.sstaf.physiology.models.api;

    requires org.slf4j;
    provides MusculatureModel with MusculatureModelImpl;
    provides Feature with MusculatureModelImpl;
    opens mil.sstaf.physiology.models.musculature to mil.sstaf.core;
}
```

A.129 SSTAF/src/features/simplePhysiologyAgent/mil.sstaf.physiology.models.respiration/build.gradle

```
ext.moduleName = 'mil.sstaf.physiology.models.respiration'
dependencies {
    implementation project(':framework:mil.sstaf.core')
    implementation project(':features:support:mil.sstaf.blackboard.api')
    implementation project(':features:ansurHandler:mil.devcom_sc.ansur.messages')
    implementation project(':features:ansurHandler:mil.devcom_sc.ansur.api')
    implementation project(':features:simplePhysiologyAgent:mil.sstaf.physiology.api')
    implementation project(':features:simplePhysiologyAgent:mil.sstaf.physiology.models.api')
}
```

A.130 SSTAF/src/features/simplePhysiologyAgent/mil.sstaf.physiology.models.respiration/src/main/java/mil/sstaf/physiology/models/respiration/RespirationModelImpl.java

```
package mil.sstaf.physiology.models.respiration;
import mil.sstaf.blackboard.api.Blackboard;
import mil.sstaf.core.features.BaseFeature;
import mil.sstaf.core.features.FeatureConfiguration;
import mil.sstaf.core.features.Requires;
import mil.sstaf.core.util.SSTAFException;
import mil.sstaf.physiology.models.api.RespirationMetrics;
import mil.sstaf.physiology.models.api.RespirationModel;
import org.apache.commons.math3.random.MersenneTwister;
public class RespirationModelImpl extends BaseFeature implements RespirationModel {
    public static final String FEATURE_NAME = "Respiration Model";
    public static final String BK_RESPIRATION_METRICS = "Respiration Metrics";
    public static final int MAJOR_VERSION = 0;
    public static final int MINOR_VERSION = 1;
    public static final int PATCH_VERSION = 0;
    @Requires
    Blackboard blackboard;
    private final MersenneTwister rng = new MersenneTwister();
    public RespirationModelImpl() {
        super(FEATURE_NAME, MAJOR_VERSION, MINOR_VERSION, PATCH_VERSION,
                false, "Just breathe!");
    }
    @Override
    public RespirationMetrics evaluate(long time_ms) {
        double spO2 = 1.0 - Math.abs(rng.nextGaussian()) * 0.5;
        var rm =  RespirationMetrics.builder()
                .spO2(spO2).timestamp_ms(time_ms).build();
        blackboard.addEntry(BK_RESPIRATION_METRICS, rm, time_ms);
        return rm;
    }
    /**
     * Initializes this {@code Provider}
     *
     * @throws SSTAFException if an error occurs.
     */
    @Override
    public void init() throws SSTAFException {
        super.init();
    }
    /**
     * Sets the configuration for this provider.
     * <p>
     * The configuration is applied when {@code init()} is invoked.
     * @param configuration
     *
     */
    @Override
    public void configure(FeatureConfiguration configuration) {
        super.configure(configuration);
        rng.setSeed(configuration.getSeed());
    }
}
```

A.131 SSTAF/src/features/simplePhysiologyAgent/mil.sstaf.physiology.models.respiration/src/main/java/module-info.java

```
import mil.sstaf.core.features.Feature;
import mil.sstaf.physiology.models.api.RespirationModel;
import mil.sstaf.physiology.models.respiration.RespirationModelImpl;
module mil.sstaf.physiology.models.respiration {
    exports mil.sstaf.physiology.models.respiration;
    requires mil.sstaf.core;
    requires mil.sstaf.physiology.models.api;

    requires org.slf4j;
    requires mil.sstaf.blackboard.api;
    provides RespirationModel with RespirationModelImpl;
    provides Feature with RespirationModelImpl;
    opens mil.sstaf.physiology.models.respiration to mil.sstaf.core;
}
```

A.132 SSTAF/src/features/simplePhysiologyAgent/mil.sstaf.physiology.models.vision/build.gradle

```
ext.moduleName = 'mil.sstaf.physiology.models.vision'
dependencies {
    implementation project(':framework:mil.sstaf.core')
    implementation project(':features:support:mil.sstaf.blackboard.api')
    implementation project(':features:ansurHandler:mil.devcom_sc.ansur.messages')
    implementation project(':features:ansurHandler:mil.devcom_sc.ansur.api')
    implementation project(':features:simplePhysiologyAgent:mil.sstaf.physiology.api')
    implementation project(':features:simplePhysiologyAgent:mil.sstaf.physiology.models.api')
}
```

A.133 SSTAF/src/features/simplePhysiologyAgent/mil.sstaf.physiology.models.vision/src/main/java/mil/sstaf/physiology/models/vision/VisionModelImpl.java

```
package mil.sstaf.physiology.models.vision;
import mil.sstaf.blackboard.api.Blackboard;
import mil.sstaf.core.features.BaseFeature;
import mil.sstaf.core.features.FeatureConfiguration;
import mil.sstaf.core.features.Requires;
import mil.sstaf.core.util.SSTAFException;
import mil.sstaf.physiology.models.api.VisionMetrics;
import mil.sstaf.physiology.models.api.VisionModel;
public class VisionModelImpl extends BaseFeature implements VisionModel {
    public static final String FEATURE_NAME = "Vision Model";
    public static final String BK_VISION_METRICS = "Vision Metrics";
    public static final int MAJOR_VERSION = 0;
    public static final int MINOR_VERSION = 1;
    public static final int PATCH_VERSION = 0;
    @Requires
    Blackboard blackboard;

    public VisionModelImpl() {
        super(FEATURE_NAME, MAJOR_VERSION, MINOR_VERSION, PATCH_VERSION,
                false, "To see");
    }
    @Override
    public VisionMetrics evaluate(long time_ms) {
        var vm = VisionMetrics.builder()
                .angularResolution_mils(2.0)
                .visionRatio(1.0)
                .timestamp_ms(time_ms)
                .build();
        blackboard.addEntry(BK_VISION_METRICS, vm, time_ms);
        return vm;
    }
    /**
     * Initializes this {@code Provider}
     *
     * @throws SSTAFException if an error occurs.
     */
    @Override
    public void init() throws SSTAFException {
        super.init();
    }
    /**
     * Sets the configuration for this provider.
     * <p>
     * The configuration is applied when {@code init()} is invoked.
     * @param configuration
     *
     */
    @Override
    public void configure(FeatureConfiguration configuration) {
        super.configure(configuration);
    }
}
```

A.134 SSTAF/src/features/simplePhysiologyAgent/mil.sstaf.physiology.models.vision/src/main/java/module-info.java

```
import mil.sstaf.core.features.Feature;
import mil.sstaf.physiology.models.api.VisionModel;
import mil.sstaf.physiology.models.vision.VisionModelImpl;
module mil.sstaf.physiology.models.vision {
    exports mil.sstaf.physiology.models.vision;
    requires mil.sstaf.core;
    requires mil.sstaf.physiology.models.api;

    requires org.slf4j;
    requires mil.sstaf.blackboard.api;
    provides VisionModel with VisionModelImpl;
    provides Feature with VisionModelImpl;
    opens mil.sstaf.physiology.models.vision to mil.sstaf.core;
}
```

A.135 SSTAF/src/features/support/mil.sstaf.blackboard.api/build.gradle

```
plugins {
    id 'java-library'
}
dependencies {
    implementation project(':framework:mil.sstaf.core')
    implementation group: 'org.apache.commons', name: 'commons-math3', version: '3.6.1'
    implementation group: 'org.slf4j', name: 'slf4j-api', version: '1.7.32'
}
```

A.136 SSTAF/src/features/support/mil.sstaf.blackboard.api/src/main/java/mil/sstaf/blackboard/api/AddEntryRequest.java

```
package mil.sstaf.blackboard.api;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.EqualsAndHashCode;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import mil.sstaf.core.entity.EntityHandle;
import mil.sstaf.core.features.HandlerContent;
/**
 * Message class for adding an item to the {@code Blackboard}
 */
@SuperBuilder
@Jacksonized
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
@EqualsAndHashCode(callSuper = true)
public class AddEntryRequest extends HandlerContent {
    public final String key;
    public final Object value;
    public final long timestamp_ms;
    public final long expiration_ms;
    /**
     * Constructor
     *
     * @param key           the key for the entry
     * @param value         the entry
     * @param timestamp_ms  the time at which the entry becomes valid. Queries before this time w
ill fail.
     * @param expiration_ms the time at which the entry expires.
     */
    public AddEntryRequest(final String key, final Object value, final long timestamp_ms, final l
ong expiration_ms) {
        super();
        this.key = key;
        this.value = value;
        this.timestamp_ms = timestamp_ms;
        this.expiration_ms = expiration_ms;
    }
    /**
     * Constructs a request from an {@code EntityHandle}
     *
     * @param handle the EntityHandle
     * @return a new AddEntryRequest
     */
    public static AddEntryRequest from(EntityHandle handle) {
        return new AddEntryRequest(handle.getPath(), handle, Blackboard.BIGBANG, Blackboard.FOREV
ER);
    }
}
```

A.137 SSTAF/src/features/support/mil.sstaf.blackboard.api/src/main/java/mil/sstaf/blackboard/api/AddEntryResponse.java

```
package mil.sstaf.blackboard.api;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.EqualsAndHashCode;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import mil.sstaf.core.features.HandlerContent;
@SuperBuilder
@Jacksonized
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
@EqualsAndHashCode(callSuper = true)
public class AddEntryResponse extends HandlerContent {
    public final int size;
    public AddEntryResponse(int size) {
        this.size = size;
    }
}
```

A.138 SSTAF/src/features/support/mil.sstaf.blackboard.api/src/main/java/mil/sstaf/blackboard/api/Blackboard.java

```
package mil.sstaf.blackboard.api;
import mil.sstaf.core.features.Handler;
import java.util.Optional;
/**
 * Simple caching mechanism that enables Features to share state information.
 * <p>
 * Values are only valid for a specific time window.
 */
public interface Blackboard extends Handler {
    long BIGBANG = Long.MIN_VALUE;
    long FOREVER = Long.MAX_VALUE;

    /**
     * Retrieves the entry associated with the specified key at the specified time.
     * <p>
     * Type matching is done using isAssignableFrom(), so subclasses will match a query
     * that uses a superclass.
     *
     * @param <T>            Generic type parameter.
     * @param key            the key for which the value is desired.
     * @param currentTime_ms the current simulation time.
     * @param type           The expected class for the value.
     * @return An genericized Optional that holds the value if it exists and is valid, Optional.e
mpty otherwise.
     */
    <T> Optional<T> getEntry(final String key, final long currentTime_ms, Class<T> type);
    /**
     * Returns an Optional containing the value associated with the specified key at the
     * specified time.
     * <p>
     * The value is returned as an Object rather than any specific type.
     *
     * @param key            the key for which the value is desired.
     * @param currentTime_ms the current simulation time.
     * @return a Optional containing the value if the key exists and has a value associated with
it and
     * the specified time is within the valid range for the value.
     */
    Optional<Object> getEntry(final String key, final long currentTime_ms);
    /**
     * Adds a non-expiring value to the Blackboard.
     *
     * @param key          the key associated with the value
     * @param value        the value to store
     * @param timestamp_ms the time that the value becomes valid
     */
    void addEntry(final String key, final Object value, final long timestamp_ms);
    /**
     * Adds a value with an expiration time to the Blackboard
     *
     * @param key           the key associated with the value
     * @param value         the value to store
     * @param timestamp_ms  the time at which the value becomes valid
     * @param expiration_ms the time at which the value expires
     */
    void addEntry(final String key, final Object value, final long timestamp_ms, final long expir
ation_ms);
    /**
     * Forcibly removes any data associated with the given key.
     *
     * @param key the key for which the data should be removed
     */
```

```
    void remove(final String key);
}
```

A.139 SSTAF/src/features/support/mil.sstaf.blackboard.api/src/main/java/mil/sstaf/blackboard/api/GetEntryRequest.java

```
package mil.sstaf.blackboard.api;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.EqualsAndHashCode;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import mil.sstaf.core.features.HandlerContent;
@SuperBuilder
@Jacksonized
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
@EqualsAndHashCode(callSuper = true)
public class GetEntryRequest extends HandlerContent {
    public final long time_ms;
    public final Class<?> type;
    public final String key;
    public GetEntryRequest(String key, Class<?> type, long time_ms) {
        super();
        this.time_ms = time_ms;
        this.type = type;
        this.key = key;
    }
}
```

A.140 SSTAF/src/features/support/mil.sstaf.blackboard.api/src/main/java/mil/sstaf/blackboard/api/GetEntryResponse.java

```
package mil.sstaf.blackboard.api;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.EqualsAndHashCode;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import mil.sstaf.core.features.HandlerContent;
@SuperBuilder
@Jacksonized
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
@EqualsAndHashCode(callSuper = true)
public class GetEntryResponse extends HandlerContent {
    public final Object value;
    public final String key;
    public final long time_ms;
    public final Class<?> type;
    public GetEntryResponse(Object value, String key, long time_ms, Class<?> type) {
        this.value = value;
        this.key = key;
        this.time_ms = time_ms;
        this.type = type;
    }
}
```

A.141 SSTAF/src/features/support/mil.sstaf.blackboard.api/src/main/java/mil/sstaf/blackboard/api/RemoveEntryRequest.java

```
package mil.sstaf.blackboard.api;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.EqualsAndHashCode;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import mil.sstaf.core.features.HandlerContent;
@SuperBuilder
@Jacksonized
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
@EqualsAndHashCode(callSuper = true)
public class RemoveEntryRequest extends HandlerContent {
    public final String key;
    public RemoveEntryRequest(String key) {
        super();
        this.key = key;
    }
}
```

A.142 SSTAF/src/features/support/mil.sstaf.blackboard.api/src/main/java/mil/sstaf/blackboard/api/RemoveEntryResponse.java

```
package mil.sstaf.blackboard.api;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.EqualsAndHashCode;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import mil.sstaf.core.features.HandlerContent;
@SuperBuilder
@Jacksonized
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
@EqualsAndHashCode(callSuper = true)
public class RemoveEntryResponse extends HandlerContent {
    public final int size;
    public RemoveEntryResponse(final int size) {
        this.size = size;
    }
}
```

A.143 SSTAF/src/features/support/mil.sstaf.blackboard.api/src/main/java/module-info.java

```
module mil.sstaf.blackboard.api {
    exports mil.sstaf.blackboard.api;
    requires transitive mil.sstaf.core;
    requires org.slf4j;
}
```

A.144 SSTAF/src/features/support/mil.sstaf.blackboard.inmem/build.gradle

```
dependencies {
    api project(':features:support:mil.sstaf.blackboard.api')
    implementation project(':framework:mil.sstaf.core')
    implementation group: 'org.slf4j', name: 'slf4j-api', version: '1.7.32'
    testImplementation project(':verification:mil.sstaftest.util')
    integrationTestImplementation project(':verification:mil.sstaftest.util')
    integrationTestRuntimeOnly project(':features:support:mil.sstaf.blackboard.inmem')
}
```

A.145 SSTAF/src/features/support/mil.sstaf.blackboard.inmem/src/integration/java/mil/sstaf/blackboard/inmem/integration/InMemBlackboardIntegrationTest.java

```
package mil.sstaf.blackboard.inmem.integration;
import mil.sstaf.blackboard.api.Blackboard;
import mil.sstaf.core.features.FeatureSpecification;
import mil.sstaftest.util.BaseFeatureIntegrationTest;
public class InMemBlackboardIntegrationTest extends BaseFeatureIntegrationTest<Blackboard> {
    /**
     * Constructor
     */
    protected InMemBlackboardIntegrationTest() {
        super(Blackboard.class, getSpec());
    }
    private static FeatureSpecification getSpec() {
        return FeatureSpecification.builder()
                .withFeatureClass(Blackboard.class)
                .featureName("Blackboard")
                .majorVersion(1)
                .minorVersion(0)
                .requireExact(true)
                .build();
    }
}
```

A.146 SSTAF/src/features/support/mil.sstaf.blackboard.inmem/src/integration/java/module-info.java

```
open module mil.sstaf.blackboard.inmem.integrationTest {
    exports mil.sstaf.blackboard.inmem.integration;
    requires mil.sstaftest.util;
    requires mil.sstaf.blackboard.api;
    requires org.junit.jupiter.api;

}
```

A.147 SSTAF/src/features/support/mil.sstaf.blackboard.inmem/src/main/java/mil/sstaf/blackboard/inmem/InMemBlackboard.java

```
package mil.sstaf.blackboard.inmem;

import mil.sstaf.blackboard.api.*;
import mil.sstaf.core.entity.Address;
import mil.sstaf.core.entity.Message;
import mil.sstaf.core.features.BaseHandler;
import mil.sstaf.core.features.HandlerContent;
import mil.sstaf.core.features.ProcessingResult;
import org.slf4j.LoggerFactory;
import org.slf4j.Logger;
import java.util.*;
/**
 * Simple in-memory Map-based implementation of the Blackboard
 */
public class InMemBlackboard extends BaseHandler implements Blackboard {
    public static final String FEATURE_NAME = "Blackboard";
    public static final int MAJOR_VERSION = 1;
    public static final int MINOR_VERSION = 0;
    public static final int PATCH_VERSION = 0;

    private static final Logger logger = LoggerFactory.getLogger(InMemBlackboard.class);
    private final Map<String, Entry> entryMap = new HashMap<>();
    public InMemBlackboard() {
        super(FEATURE_NAME, MAJOR_VERSION, MINOR_VERSION, PATCH_VERSION, false,
                "Simple in-memory implementation of a Blackboard");
    }
    /**
     * Retrieves the entry associated with the specified key at the specified time.
     * <p>
     * Type matching is done using isAssignableFrom(), so subclasses will match a query
     * that uses a superclass.
     *
     * @param <T>            Generic type parameter.
     * @param key            the key for which the value is desired.
     * @param currentTime_ms the current simulation time.
     * @param type           The expected class for the value.
     * @return An genericized Optional that holds the value if it exists and is valid, Optional.e
mpty otherwise.
     */
    public <T> Optional<T> getEntry(final String key, final long currentTime_ms, Class<T> type) {
        Objects.requireNonNull(key, "Blackboard query may not use a null key");
        Objects.requireNonNull(type, "Blackboard query may not specify a null type");
        if (logger.isTraceEnabled()) {
            logger.trace("Getting {}/{} valid at {}", key, type.getName(), currentTime_ms);
        }
        Optional<Object> optValue = getEntry(key, currentTime_ms);
        if (optValue.isPresent()) {
            Object val = optValue.get();
            if (type.isAssignableFrom(val.getClass())) {
                if (logger.isDebugEnabled()) {
                    logger.debug("Key '{}', returning '{}'", key, val);
                }
                return Optional.of(type.cast(val));
            }
        }
        if (logger.isDebugEnabled()) {
            logger.debug("Key '{}' not found, returning Optional.empty", key);
        }
        return Optional.empty();
    }
```

```
    /**
     * Returns an Optional containing the value associated with the specified key at the
     * specified time.
     * <p>
     * The value is returned as an Object rather than any specific type.
     *
     * @param key            the key for which the value is desired.
     * @param currentTime_ms the current simulation time.
     * @return a Optional containing the value if the key exists and has a value associated with
it and
     * the specified time is within the valid range for the value.
     */
    public Optional<Object> getEntry(final String key, final long currentTime_ms) {
        Object e1 = internalGetEntry(key, currentTime_ms);
        return Optional.ofNullable(e1);
    }
    /**
     * Internal method for querying the Blackboard.
     *
     * @param key            the key for which the value is desired.
     * @param currentTime_ms the current simulation time.
     * @return the value if found and valid, null otherwise.
     */
    private Object internalGetEntry(String key, long currentTime_ms) {
        Objects.requireNonNull(key, "Blackboard query may not use a null key");
        if (logger.isTraceEnabled()) {
            logger.trace("Getting {} valid at {}", key, currentTime_ms);
        }
        Entry e = entryMap.get(key);
        if (e != null) {
            if (logger.isTraceEnabled()) {
                logger.trace("Got entry");
            }
            if (currentTime_ms >= e.timestamp_ms) {
                if (currentTime_ms <= e.expiration_ms) {
                    if (logger.isDebugEnabled()) {
                        logger.debug("Entry is valid, returning {}", e.value);
                    }
                    return e.value;
                } else {
                    if (logger.isTraceEnabled()) {
                        logger.trace("Entry is expired, removing it");
                    }
                    entryMap.remove(key);
                }
            } else {
                if (logger.isTraceEnabled()) {
                    logger.trace("currentTime {} < entry timestamp {}", currentTime_ms, e.timesta
mp_ms);
                }
            }
        }
        return null;
    }
    /**
     * Adds a non-expiring value to the Blackboard.
     *
     * @param key          the key associated with the value
     * @param value        the value to store
     * @param timestamp_ms the time that the value becomes valid
     */
    public void addEntry(final String key, final Object value, final long timestamp_ms) {
        addEntry(key, value, timestamp_ms, FOREVER);
    }
    /**
     * Adds a value with an expiration time to the Blackboard
     *
     * @param key           the key associated with the value
```

```
     * @param value         the value to store
     * @param timestamp_ms  the time at which the value becomes valid
     * @param expiration_ms the time at which the value expires
     */
    public void addEntry(final String key, final Object value, final long timestamp_ms, final lon
g expiration_ms) {
        Objects.requireNonNull(key, "Blackboard entry may not have a null key");
        Objects.requireNonNull(value, "Blackboard entry may not have a null value");
        if (logger.isDebugEnabled()) {
            logger.debug("Adding '{}':'{}'; valid {} to  {}",
                    key, value, timestamp_ms, expiration_ms);
        }
        entryMap.put(key, new Entry(value, timestamp_ms, expiration_ms));
    }
    /**
     * Forcibly removes any data associated with the given key.
     *
     * @param key the key for which the data should be removed
     */
    public void remove(final String key) {
        entryMap.remove(key);
    }
    /**
     * Returns the map that holds the entries.
     * <p>
     * Package-visible for testing.
     *
     * @return an {@code unmodifiableMap} of the entry map;
     */
    Map<String, Entry> getEntryMap() {
        return Collections.unmodifiableMap(entryMap);
    }
    @Override
    public List<Class<? extends HandlerContent>> contentHandled() {
        return List.of(AddEntryRequest.class, GetEntryRequest.class, RemoveEntryRequest.class);
    }
    @Override
    public ProcessingResult process(HandlerContent arg, long scheduledTime_ms, long currentTime_m
s, Address from, long id, Address respondTo) {
        Message output = null;
        if (arg instanceof GetEntryRequest) {
            GetEntryRequest message = (GetEntryRequest) arg;
            Object value = internalGetEntry(message.key, message.time_ms);
            GetEntryResponse response = new GetEntryResponse(value, message.key, message.time_ms,
message.type);
            output = buildNormalResponse(response, id, respondTo);
        } else if (arg instanceof RemoveEntryRequest) {
            RemoveEntryRequest rer = (RemoveEntryRequest) arg;
            remove(rer.key);
            RemoveEntryResponse response = new RemoveEntryResponse(entryMap.size());
            output = buildNormalResponse(response, id, respondTo);
        } else if (arg instanceof AddEntryRequest) {
            AddEntryRequest aer = (AddEntryRequest) arg;
            addEntry(aer.key, aer.value, aer.timestamp_ms, aer.expiration_ms);
            AddEntryResponse response = new AddEntryResponse(entryMap.size());
            output = buildNormalResponse(response, id, respondTo);
        }
        if (output == null) {
            return buildUnsupportedMessageResponse(arg, id, respondTo, new UnsupportedOperationEx
ception());
        } else {
            return ProcessingResult.of(output);
        }
    }
    static class Entry {
        final long timestamp_ms;
        final long expiration_ms;
        final Object value;
```

```
        Entry(Object value, long timestamp_ms, long expiration_ms) {
            this.timestamp_ms = timestamp_ms;
            this.expiration_ms = expiration_ms;
            this.value = value;
        }
    }
}
```

A.148 SSTAF/src/features/support/mil.sstaf.blackboard.inmem/src/main/java/module-info.java

```
import mil.sstaf.blackboard.api.Blackboard;
import mil.sstaf.blackboard.inmem.InMemBlackboard;
import mil.sstaf.core.features.Feature;
import mil.sstaf.core.features.Handler;
module mil.sstaf.blackboard.inmem {
    exports mil.sstaf.blackboard.inmem;
    requires mil.sstaf.blackboard.api;

    requires org.slf4j;
    provides Feature with InMemBlackboard;
    provides Handler with InMemBlackboard;
    provides Blackboard with InMemBlackboard;
    opens mil.sstaf.blackboard.inmem to mil.sstaf.core;
}
```

A.149 SSTAF/src/features/support/mil.sstaf.blackboard.inmem/src/test/java/mil/sstaf/blackboard/inmem/BlackboardTest.java

```
package mil.sstaf.blackboard.inmem;
import lombok.experimental.SuperBuilder;
import mil.sstaf.blackboard.api.*;
import mil.sstaf.core.entity.*;
import mil.sstaf.core.features.ExceptionCommand;
import mil.sstaf.core.features.Handler;
import mil.sstaf.core.features.ProcessingResult;
import mil.sstaf.core.entity.Message;
import mil.sstaf.core.util.Injector;
import mil.sstaftest.util.BaseHandlerTest;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Nested;
import org.junit.jupiter.api.Test;
import java.util.List;
import java.util.Optional;
import static org.junit.jupiter.api.Assertions.*;
class BlackboardTest extends BaseHandlerTest<InMemBlackboard> {
    static {
        preloadedClasses = List.of();
        String value = "I am the answer";
        String key = "daKey";
        AddEntryRequest aer = new AddEntryRequest(key, value, 1, Blackboard.FOREVER);
        GetEntryRequest ger = new GetEntryRequest(key, String.class, 1001000);
        RemoveEntryRequest rer = new RemoveEntryRequest(key);
        sampleMessages = List.of(aer, ger, rer);
    }

    long toMillis(double s) {
        return Math.round(s * 1000);
    }
    @Override
    protected InMemBlackboard buildFeature() {
        return new InMemBlackboard();
    }
    @SuperBuilder
    static class FakeEntity extends BaseEntity {
        @Override
        public String getPath() {
            return "";
        }
    }
    @Nested
    public class BlackboardContractTests {
        @Test
        @DisplayName("An entry with an explicit expiration time works as expected")
        void entryWithExpirationWorksAsExpected() {
            String value = "I am the answer";
            String key = "daKey";
            InMemBlackboard blackboard = new InMemBlackboard();
            blackboard.addEntry(key, value, toMillis(1.0), toMillis(10.0));
            assertEquals(1, blackboard.getEntryMap().size());
            assertTrue(blackboard.getEntryMap().containsKey(key));
            Optional<String> response0 = blackboard.getEntry(key, toMillis(0.5), String.class);
            assertFalse(response0.isPresent(), "Entry is not valid yet");
            Optional<String> response1 = blackboard.getEntry(key, toMillis(7.5), String.class);
            assertTrue(response1.isPresent());
            assertEquals(value, response1.get());
            Optional<String> response2 = blackboard.getEntry(key, 10000, String.class);
            assertTrue(response2.isPresent());
            assertEquals(value, response2.get());
            Optional<String> response3 = blackboard.getEntry(key, 15000, String.class);
            assertFalse(response3.isPresent(), "Entry has expired and should be gone");
            assertEquals(0, blackboard.getEntryMap().size());
```

```
            assertFalse(blackboard.getEntryMap().containsKey(key));
        }
        @Test
        void entryWithIndefiniteExpirationWorksAsExpected() {
            String value = "I am the answer";
            String key = "daKey";
            InMemBlackboard blackboard = new InMemBlackboard();
            blackboard.addEntry(key, value, 1000);
            assertEquals(1, blackboard.getEntryMap().size());
            assertTrue(blackboard.getEntryMap().containsKey(key));
            Optional<String> response1 = blackboard.getEntry(key, 7500, String.class);
            assertTrue(response1.isPresent());
            assertEquals(value, response1.get());
            Optional<String> response2 = blackboard.getEntry(key, 10000, String.class);
            assertTrue(response2.isPresent());
            assertEquals(value, response2.get());
            Optional<String> response3 = blackboard.getEntry(key, 15000, String.class);
            assertTrue(response3.isPresent());
            assertEquals(value, response3.get());
            Optional<String> response4 = blackboard.getEntry(key, Blackboard.FOREVER, String.clas
s);
            assertTrue(response4.isPresent());
            assertEquals(value, response4.get());
        }
        @Test
        void updatingTheValuesWorksAsExpected() {
            String value = "I am the answer";
            String value2 = "No I am";
            String key = "daKey";
            InMemBlackboard blackboard = new InMemBlackboard();
            blackboard.addEntry(key, value, toMillis(1.0));
            assertEquals(1, blackboard.getEntryMap().size());
            assertTrue(blackboard.getEntryMap().containsKey(key));
            Optional<String> response1 = blackboard.getEntry(key, 7500, String.class);
            assertTrue(response1.isPresent());
            assertEquals(value, response1.get());
            Optional<String> response2 = blackboard.getEntry(key, 10000, String.class);
            assertTrue(response2.isPresent());
            assertEquals(value, response2.get());
            blackboard.addEntry(key, value2, 23000, 50000);
            Optional<String> response3 = blackboard.getEntry(key, 15000, String.class);
            assertFalse(response3.isPresent(),
                    "Old entry has been replaced, but new entry should not be valid yet");
            Optional<String> response4 = blackboard.getEntry(key, 49900, String.class);
            assertTrue(response4.isPresent());
            assertEquals(value2, response4.get());
        }
        @Test
        void nullKeyWithExpirationThrows() {
            String value = "I am the answer";
            Blackboard blackboard = new InMemBlackboard();
            assertThrows(NullPointerException.class,
                    () -> blackboard.addEntry(null, value, 1000, 10000));
        }
        @Test
        void nullKeyWithoutExpirationThrows() {
            String value = "I am the answer";
            Blackboard blackboard = new InMemBlackboard();
            assertThrows(NullPointerException.class, () -> blackboard.addEntry(null, value, 1000)
);
        }
        @Test
        void nullValueWithExpirationThrows() {
            String key = "daKey";
            Blackboard blackboard = new InMemBlackboard();
            assertThrows(NullPointerException.class,
                    () -> blackboard.addEntry(key, null, 1000, 10000));
        }
```

```
        @Test
        void nullValueWithoutExpirationThrows() {
            String key = "daKey";
            Blackboard blackboard = new InMemBlackboard();
            assertThrows(NullPointerException.class,
                    () -> blackboard.addEntry(key, null, 1000));
        }
        @Test
        void getWithNullKeyThrows1() {
            String value = "I am the answer";
            String key = "daKey";
            Blackboard blackboard = new InMemBlackboard();
            blackboard.addEntry(key, value, 1000);
            assertThrows(NullPointerException.class, () ->
                    blackboard.getEntry(null, 10000));
        }
        @Test
        void getWithNullKeyThrows2() {
            String value = "I am the answer";
            String key = "daKey";
            Blackboard blackboard = new InMemBlackboard();
            blackboard.addEntry(key, value, 1000);
            assertThrows(NullPointerException.class,
                    () -> blackboard.getEntry(null, 10000, String.class));
        }
        @Test
        void getWithNullClassThrows() {
            String value = "I am the answer";
            String key = "daKey";
            Blackboard blackboard = new InMemBlackboard();
            blackboard.addEntry(key, value, 1000);
            assertThrows(NullPointerException.class,
                    () -> blackboard.getEntry(key, 10000, null));
        }
        @Test
        void removeWorks() {
            String value = "I am the answer";
            String key = "daKey";
            InMemBlackboard blackboard = new InMemBlackboard();
            blackboard.addEntry(key, value, 1000);
            assertEquals(1, blackboard.getEntryMap().size());
            blackboard.remove("notDaKey");
            assertEquals(1, blackboard.getEntryMap().size());
            blackboard.remove(key);
            assertEquals(0, blackboard.getEntryMap().size());
        }
        @Test
        void removeWithNullClassDoesNotThrow() {
            String value = "I am the answer";
            String key = "daKey";
            Blackboard blackboard = new InMemBlackboard();
            blackboard.addEntry(key, value, 1000);
            assertDoesNotThrow(() -> blackboard.remove(null));
        }

        @Test
        void handlerIntefaceWorksAsExpected() {
            Handler handler = new InMemBlackboard();
            FakeEntity fakeEntity =  FakeEntity.builder().build();
            EntityHandle entityHandle = fakeEntity.getHandle();
            Injector.inject(handler, entityHandle);
            assertEquals("Blackboard", handler.getName());
            assertEquals(1, handler.getMajorVersion());
            assertEquals(0, handler.getMinorVersion());
            assertEquals(0, handler.getPatchVersion());
            assertNotNull(handler.getDescription());
            String value = "I am the answer";
            String key = "daKey";
```

```
            Address address = Address.makeAddress(entityHandle, "bob");
            AddEntryRequest aer = new AddEntryRequest(key, value, 1, Blackboard.FOREVER);
            ProcessingResult pr1 = handler.process(aer, 100000, 100000, address, 1, address);
            Message mr1 = pr1.messages.get(0);
            Object contents = mr1.getContent();
            assertTrue(contents instanceof AddEntryResponse);
            AddEntryResponse response = (AddEntryResponse) contents;
            assertEquals(1, response.size);
            GetEntryRequest ger = new GetEntryRequest(key, String.class, 1001000);
            ProcessingResult pr2 = handler.process(ger, 100000, 100000, address, 1, address);
            Message mr2 = pr2.messages.get(0);
            Object contents2 = mr2.getContent();
            assertTrue(contents2 instanceof GetEntryResponse);
            GetEntryResponse response2 = (GetEntryResponse) contents2;
            assertEquals(value, response2.value);
            RemoveEntryRequest rer = new RemoveEntryRequest(key);
            ProcessingResult pr3 = handler.process(rer, 100000, 100000, address, 1, address);
            Message mr3 = pr3.messages.get(0);
            Object content3 = mr3.getContent();
            assertTrue(content3 instanceof RemoveEntryResponse);
            RemoveEntryResponse response3 = (RemoveEntryResponse) content3;
            assertEquals(0, response3.size);
            ProcessingResult pr4 = handler.process(ExceptionCommand.builder().build(), 100000, 10
0000, address, 1, address);
            Message mr4 = pr4.messages.get(0);
            assertTrue(mr4 instanceof ErrorResponse);
        }
    }
}
```

A.150 SSTAF/src/features/support/mil.sstaf.blackboard.inmem/src/test/resources/log4j2-test.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="INFO">
    <Appenders>
        <Console name="Console" target="SYSTEM_OUT">
            <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"/>
        </Console>
    </Appenders>
    <Loggers>
        <Root level="debug">
            <AppenderRef ref="Console"/>
        </Root>
    </Loggers>
</Configuration>
```

A.151 SSTAF/src/features/support/mil.sstaf.telemetry/build.gradle

```
plugins {
    id 'java-library'
}
ext.moduleName = 'mil.sstaf.telemetry'
dependencies {
    implementation project(':framework:mil.sstaf.core')
    implementation project(':features:support:mil.sstaf.blackboard.api')
    implementation group: 'org.slf4j', name: 'slf4j-api', version: '1.7.32'
    testImplementation project(':verification:mil.sstaftest.util')
    testRuntimeOnly project(':features:support:mil.sstaf.blackboard.inmem')
}
testing {
    suites {
        integrationTest {
            dependencies {
                implementation project(':framework:mil.sstaf.core')
                implementation project(':features:support:mil.sstaf.blackboard.api')
                implementation project(':verification:mil.sstaftest.util')
                implementation project(':testFeatures:integration:mil.sstaftest.simplemock')
                implementation project(':testFeatures:integration:mil.sstaftest.jamesbond')
                implementation project(':testFeatures:integration:mil.sstaftest.alpha')
                implementation project(':testFeatures:integration:mil.sstaftest.bravo')
                implementation project(':testFeatures:integration:mil.sstaftest.charlie')
                implementation project(':testFeatures:integration:mil.sstaftest.delta')
                implementation project(':testFeatures:integration:mil.sstaftest.echo')
                runtimeOnly project(':testFeatures:support:mil.sstaftest.mocks.handler1')
                runtimeOnly project(':testFeatures:support:mil.sstaftest.mocks.agent1')
                runtimeOnly project(':testFeatures:support:mil.sstaftest.mocks.pinky')
                runtimeOnly project(':features:support:mil.sstaf.blackboard.inmem')
            }
        }
    }
}
```

A.152 SSTAF/src/features/support/mil.sstaf.telemetry/src/integrationTest/java/mil/sstaf/telemetry/integration/TelemetryIntegrationTest.java

```
package mil.sstaf.telemetry.integration;
import mil.sstaf.blackboard.api.Blackboard;
import mil.sstaf.core.features.Agent;
import mil.sstaf.core.features.FeatureSpecification;
import mil.sstaf.telemetry.TelemetryAgent;
import mil.sstaf.telemetry.TelemetryConfiguration;
import mil.sstaftest.util.BaseFeatureIntegrationTest;
public class TelemetryIntegrationTest extends BaseFeatureIntegrationTest<Agent, TelemetryConfigur
ation> {
    /**
     * Constructor
     */
    protected TelemetryIntegrationTest() {
        super(Agent.class, getSpec());
    }
    private static FeatureSpecification getSpec() {
        return FeatureSpecification.builder()
                .featureClass(TelemetryAgent.class)
                .featureName("Telemetry Agent")
                .majorVersion(1)
                .minorVersion(0)
                .requireExact(true)
                .build();
    }
}
```

A.153 SSTAF/src/features/support/mil.sstaf.telemetry/src/integrationTest/java/module-info.java

```
open module mil.sstaf.telemetry.integrationTest{
    exports mil.sstaf.telemetry.integration;
    requires mil.sstaftest.util;
    requires mil.sstaf.telemetry;
    requires mil.sstaf.blackboard.api;
    requires org.junit.jupiter.api;

}
```

A.154 SSTAF/src/features/support/mil.sstaf.telemetry/src/main/java/mil/sstaf/telemetry/TelemetryAgent.java

```
package mil.sstaf.telemetry;

import mil.sstaf.blackboard.api.Blackboard;
import mil.sstaf.core.entity.Address;
import mil.sstaf.core.entity.Entity;
import mil.sstaf.core.entity.EntityHandle;
import mil.sstaf.core.features.*;
import mil.sstaf.core.state.State;
import mil.sstaf.core.state.StateProperty;
import mil.sstaf.core.util.SSTAFException;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.PrintStream;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class TelemetryAgent extends BaseAgent {
    public static final String FEATURE_NAME = "Telemetry Agent";
    public static final int MAJOR_VERSION = 1;
    public static final int MINOR_VERSION = 0;
    public static final int PATCH_VERSION = 0;

    public static final String CK_STATES_LIST = "statesToRecord";
    public static final String PROP_TELEMETRY_OUTPUT_DIR = "mil.sstaf.telemetry.outputDir";
    private static final Logger logger = LoggerFactory.getLogger(TelemetryAgent.class);
    private final Map<String, StateWriter> writerMap = new HashMap<>();
    private final List<String> stateKeys = new ArrayList<>();

    @Requires
    Blackboard blackboard;

    /**
     * Constructor
     */
    public TelemetryAgent() {
        super(FEATURE_NAME, MAJOR_VERSION, MINOR_VERSION, PATCH_VERSION,
                true, "CSV telemetry writer");
    }

    /**
     * Sets the configuration for this provider.
     * <p>
     * The configuration is applied when {@code init()} is invoked.
     *
     * @param configuration the configuration
     */
    @Override
    public void configure(FeatureConfiguration configuration) {
        super.configure(configuration);
        if (configuration instanceof TelemetryConfiguration) {
            TelemetryConfiguration tc = (TelemetryConfiguration) configuration;
            if (tc.getStateKeys() != null) {
                this.stateKeys.addAll(tc.getStateKeys());
            }
        }
```

```
    }

    /**
     * {@inheritDoc}
     */
    @Override
    public void init() {
        super.init();
        try {
            File outputDir = makeOutputDir();
            logger.info("Writing telemetry for '" + ownerHandle.getPath() + "' to " + outputDir.g
etPath());
            stateKeys.forEach(key -> writerMap.put(key, new StateWriter(key, outputDir)));
        } catch (FileNotFoundException e) {
            throw new SSTAFException(e);
        }
    }

    private String filePathFromEntityPath(EntityHandle entityHandle) {
        String[] elements = entityHandle.getPath().split(Entity.ENTITY_PATH_DELIMITER);
        StringBuilder stringBuilder = new StringBuilder();
        for (String element : elements) {
            stringBuilder.append(element).append(File.separator);
        }
        return stringBuilder.toString();
    }

    /**
     *
     */
    private File makeOutputDir() throws FileNotFoundException {
        String logDirPath = System.getProperty(PROP_TELEMETRY_OUTPUT_DIR, "sstafTelemetry");
        String fullPath = logDirPath + File.separator + filePathFromEntityPath(ownerHandle);
        File outputDir = new File(fullPath);
        if (!outputDir.exists()) {
            if (!outputDir.mkdirs()) {
                throw new FileNotFoundException("Can't make telemetry output directory " + output
Dir);
            }
        }
        return outputDir;
    }

    /**
     * Activate the Agent to perform a function at the specified time.
     *
     * @param currentTime_ms the simulation time.
     * @return a {@code ProcessingResult} containing internal and external {@code Message}s
     */
    @Override
    public ProcessingResult tick(long currentTime_ms) {
        if (logger.isDebugEnabled()) {
            logger.debug("{}:{} - Ticking at {}", ownerHandle.getPath(), featureName, currentTime
_ms);
        }
        writerMap.values().forEach(writer -> writer.writeValues(currentTime_ms));
        return ProcessingResult.empty();
    }

    /**
     * Provides a {@code List} of all of the message content {@code Class}es to which this
     * {@code Handle} responds.
     * <p>
     * Handlers can respond to multiple message classes. Only one Handler may be associated with
     * a message class.
     *
     * @return a {@code List} of {@code Class} objects.
     */
```

```
    @Override
    public List<Class<? extends HandlerContent>> contentHandled() {
        return List.of();
    }

    /**
     * Performs the processing associated with the given argument at the specified time.
     * <p>
     * Both the scheduled time and current simulation time are reported. For an {@code EntityEven
t}
     * the scheduled time value will be less than or equal the current simulation time. For an
     * {@code EntityAction} scheduled time will be the same as the current time. Providing both t
imes
     * enables the Handler to adjust for any time difference between when an event was intended t
o occur
     * and when it was processed.
     *
     * @param arg              the message content
     * @param scheduledTime_ms the time for which an event was scheduled.
     * @param currentTime_ms   the current simulation time
     * @param from             the sources of the {@code Message}
     * @param id               the message sequence numner
     * @param respondTo        to where to send the results.
     * @return a {@code ProcessingResult} that contains the results of the processing
     */
    @Override
    public ProcessingResult process(HandlerContent arg, long scheduledTime_ms, long currentTime_m
s, Address from, long id, Address respondTo) {
        return buildUnsupportedMessageResponse(arg, id, respondTo, new UnsupportedOperationExcept
ion());
    }

    /**
     * Accessor for tests
     *
     * @return the Blackboard
     */
    Blackboard getBlackboard() {
        return blackboard;
    }

    class StateWriter {
        private final String key;
        private final List<Method> methods = new ArrayList<>();
        private final List<String> headers = new ArrayList<>();
        private PrintStream printStream;
        private boolean configured = false;
        private Class<?> configuredClass;

        private StateWriter(String key, File parentDir) {
            this.key = key;
            String path = parentDir.getPath() + File.separator + key + ".csv";
            File outputFile = new File(path);
            try {
                this.printStream = new PrintStream(outputFile);
            } catch (FileNotFoundException e) {
                this.printStream = System.out;
            }
            if (logger.isTraceEnabled()) {
                logger.trace("{}:{} created writer for key '{}'", ownerHandle.getPath(), featureN
ame, key);
            }
        }

        private void getMethodWithStateProperty(final Class<?> target) {
            if (logger.isDebugEnabled()) {
                logger.debug("{} - Scanning methods for StateProperty annotations in ", target.ge
tName());
```

```
            }
            if (Object.class.equals(target)) {
                if (logger.isTraceEnabled()) {
                    logger.trace("Ignoring Object.class ");
                }
                return;
            } else {
                getMethodWithStateProperty(target.getSuperclass());
                if (logger.isTraceEnabled()) {
                    logger.trace("{} - returned from superclass ", target.getSuperclass().getName
());
                }
            }
            for (Method method : target.getDeclaredMethods()) {
                if (logger.isTraceEnabled()) {
                    logger.trace("{} - Processing field {}", target.getName(), method.getName());
                }
                if (method.isAnnotationPresent(StateProperty.class)) {
                    if (logger.isDebugEnabled()) {
                        logger.debug("{} - Registering field {}", target.getName(), method.getNam
e());
                    }
                    methods.add(method);
                    StateProperty sp = method.getAnnotation(StateProperty.class);
                    String headerName = sp.headerLabel();
                    if (headerName.length() == 0) {
                        headerName = method.getName();
                    }
                    headers.add(headerName);
                } else {
                    if (logger.isTraceEnabled()) {
                        logger.trace("{} - Field {} is not annotated", target.getName(), method.g
etName());
                    }
                }
            }
            if (logger.isDebugEnabled()) {
                logger.debug("{} - Found {} @StateProperty annotations, {}", target.getName(),
                        methods.size(), methods);
            }
        }

        void writeHeader() {
            if (logger.isTraceEnabled()) {
                logger.trace("{}:{} writing header for key '{}'", ownerHandle.getPath(), featureN
ame, key);
            }
            printStream.print("\"Time\",");
            for (String header : headers) {
                String x = '"' + header + '"' + ',';
                printStream.print(x);
            }
            printStream.println();
            printStream.flush();
        }

        void writeValues(long currentTime_ms) {
            if (logger.isDebugEnabled()) {
                logger.debug("{}:{} writing values for key '{}'", ownerHandle.getPath(), featureN
ame, key);
            }
            blackboard.getEntry(key, currentTime_ms, State.class)
                    .ifPresentOrElse(
                            state -> writeValues(currentTime_ms, state),
                            () -> {
                                if (logger.isDebugEnabled()) {
                                    logger.debug("{}:{} No entry for key '{}'", ownerHandle.getPa
th(),
```

```
                                        featureName, key);
                                }
                            });
        }

        void writeValues(long currentTime_ms, State target) {
            if (configured) {
                if (!configuredClass.isAssignableFrom(target.getClass())) {
                    throw new SSTAFException("Record type changed");
                }
            } else {
                getMethodWithStateProperty(target.getClass());
                writeHeader();
                configured = true;
                configuredClass = target.getClass();
            }

            printStream.print(currentTime_ms);
            printStream.print(", ");

            for (Method method : methods) {
                boolean accessible = method.canAccess(target);
                if (!accessible) {
                    method.setAccessible(true);
                }
                try {
                    Class<?> cl = method.getReturnType();
                    if (int.class.equals(cl)) {
                        int val = (Integer) method.invoke(target);
                        printStream.print(val);
                    } else if (double.class.equals(cl)) {
                        double val = (Double) method.invoke(target);
                        printStream.print(val);
                    } else if (boolean.class.equals(cl)) {
                        boolean val = (Boolean) method.invoke(target);
                        printStream.print(val);
                    } else {
                        Object val = method.invoke(target);
                        String output = '"' + val.toString() + '"';
                        printStream.print(output);
                    }
                    printStream.print(", ");
                } catch (IllegalAccessException | InvocationTargetException e) {
                    e.printStackTrace();
                }
            }
            printStream.println();
            printStream.flush();
        }
    }
}
```

A.155 SSTAF/src/features/support/mil.sstaf.telemetry/src/main/java/mil/sstaf/telemetry/TelemetryConfiguration.java

```
package mil.sstaf.telemetry;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.Getter;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import mil.sstaf.core.features.FeatureConfiguration;
import java.util.List;
@Jacksonized
@SuperBuilder
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
public class TelemetryConfiguration extends FeatureConfiguration {
    @Getter
    private List<String> stateKeys;
}
```

A.156 SSTAF/src/features/support/mil.sstaf.telemetry/src/main/java/module-info.java

```
import mil.sstaf.core.features.Agent;
import mil.sstaf.core.features.Feature;
import mil.sstaf.core.features.Handler;
import mil.sstaf.telemetry.TelemetryAgent;
module mil.sstaf.telemetry {
    exports mil.sstaf.telemetry;
    requires mil.sstaf.core;
    requires org.slf4j;

    requires mil.sstaf.blackboard.api;
    provides Feature with TelemetryAgent;
    provides Handler with TelemetryAgent;
    provides Agent with TelemetryAgent;
    opens mil.sstaf.telemetry to mil.sstaf.core, com.fasterxml.jackson.databind;
}
```

A.157 SSTAF/src/features/support/mil.sstaf.telemetry/src/test/java/mil/sstaf/telemetry/TelemetryAgentTest.java

```
package mil.sstaf.telemetry;
import lombok.Data;
import lombok.EqualsAndHashCode;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import mil.sstaf.core.state.LabeledState;
import mil.sstaf.core.state.StateProperty;
import mil.sstaftest.util.BaseAgentTest;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Nested;
import org.junit.jupiter.api.Test;
import java.io.File;
import java.util.List;
import java.util.Random;
import static org.junit.jupiter.api.Assertions.*;

public class TelemetryAgentTest extends BaseAgentTest<TelemetryAgent> {
    public static final String TELEMETRY_DIR = "build" + File.separator + "tmp" + File.separator
+ "telemetryDir";
    static {
        preloadedClasses = List.of("mil.sstaf.blackboard.inmem.InMemBlackboard");
        TelemetryConfiguration tc = TelemetryConfiguration.builder()
                .stateKeys(List.of("Chuck", "Sarah")).build();
        setDefaultConfiguration("Telemetry Agent", tc);
        System.setProperty(TelemetryAgent.PROP_TELEMETRY_OUTPUT_DIR, TELEMETRY_DIR);
    }
    public TelemetryAgent buildFeature() {
        return new TelemetryAgent();
    }
    @BeforeEach
    void setup() {
        feature = setupFeature();
    }
    @Data
    @EqualsAndHashCode(callSuper = true)
    @SuperBuilder
    @Jacksonized
    static class ChuckState extends LabeledState {
        final double x;
        final boolean y;
        final int z;
        @StateProperty(headerLabel = "EX")
        public double getX() {
            return x;
        }
        @StateProperty(headerLabel = "Why?")
        public boolean isY() {
            return y;
        }
        @StateProperty(headerLabel = "ZEE!")
        public int getZ() {
            return z;
        }
    }
    @Data
    @EqualsAndHashCode(callSuper = true)
    @SuperBuilder
    @Jacksonized
    static class SarahState extends LabeledState {
        final double a;
        final boolean b;
```

```
        @StateProperty(headerLabel = "Alpha")
        public double getA() {
            return a;
        }
        @StateProperty(headerLabel = "Bravo")
        public boolean isB() {
            return b;
        }
    }
    @Nested
    @DisplayName("Check the specific requirements for 'TelemetryAgent'")
    public class TelemetryAgentTests {
        @Test
        @DisplayName("Confirm that the TelemetryAgent can write CSV files in the specified direct
ory")
        void simpleTest() {
            assertNotNull(feature);
            assertDoesNotThrow(feature::init);
            Random random = new Random();
            for (int i = 0; i < 100.0; ++i) {
                ChuckState chuck = ChuckState.builder().x(random.nextDouble()).y(random.nextBoole
an())
                        .z(random.nextInt()).build();
                SarahState sarah = SarahState.builder().a(random.nextDouble()).b(random.nextBoole
an()).build();
                feature.getBlackboard().addEntry("Chuck", chuck, 0);
                feature.getBlackboard().addEntry("Sarah", sarah, 0);
                feature.tick(i);
            }
            String chuckPath = TELEMETRY_DIR + File.separator + "Dummy" + File.separator + "Chuck
.csv";
            File chuckFile = new File(chuckPath);
            assertTrue(chuckFile.exists());
            assertTrue(chuckFile.isFile());
            String sarahPath = TELEMETRY_DIR + File.separator + "Dummy" + File.separator + "Sarah
.csv";
            File sarahFile = new File(sarahPath);
            assertTrue(sarahFile.exists());
            assertTrue(sarahFile.isFile());
        }
    }
}
```

A.158 SSTAF/src/features/support/mil.sstaf.telemetry/src/test/resources/log4j2-test.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="INFO">
    <Appenders>
        <Console name="Console" target="SYSTEM_OUT">
            <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"/>
        </Console>
    </Appenders>
    <Loggers>
        <Logger name="mil.sstaf.telemetry.TelemetryAgent" level="trace" additivity="false">
            <AppenderRef ref="Console"/>
        </Logger>
        <!--
        <Logger name="mil.sstaf.core.features.Resolver" level="trace" additivity="false">
            <AppenderRef ref="Console"/>
        </Logger>
        <Logger name="mil.sstaf.core.features.FeatureLoader" level="trace" additivity="false">
            <AppenderRef ref="Console"/>
        </Logger>
        -->
        <Root level="debug">
            <AppenderRef ref="Console"/>
        </Root>
    </Loggers>
</Configuration>
```

A.159 SSTAF/src/framework/mil.sstaf.analyzer/build.gradle

```
dependencies {
    implementation project(':framework:mil.sstaf.core')
    implementation project(':framework:mil.sstaf.session')
    implementation group: 'org.apache.commons', name: 'commons-math3', version: '3.6.1'
    implementation group: 'org.slf4j', name: 'slf4j-api', version: '1.7.32'
}
testing {
    suites {
        test {
            dependencies {
                implementation project(':framework:mil.sstaf.core')
                implementation project(':framework:mil.sstaf.session')
                implementation project(':testFeatures:integration:mil.sstaftest.simplemock')
                implementation project(':testFeatures:integration:mil.sstaftest.jamesbond')
                implementation project(':testFeatures:integration:mil.sstaftest.alpha')
                implementation project(':testFeatures:integration:mil.sstaftest.bravo')
                implementation project(':testFeatures:integration:mil.sstaftest.charlie')
                implementation project(':testFeatures:integration:mil.sstaftest.delta')
                implementation project(':testFeatures:integration:mil.sstaftest.echo')
                runtimeOnly project(':testFeatures:support:mil.sstaftest.mocks.handler1')
                runtimeOnly project(':testFeatures:support:mil.sstaftest.mocks.agent1')
                runtimeOnly project(':testFeatures:support:mil.sstaftest.mocks.pinky')
                runtimeOnly project(':features:ansurHandler:mil.devcom_sc.ansur.api')
                runtimeOnly project(':features:ansurHandler:mil.devcom_sc.ansur.handler')
                runtimeOnly project(':features:ansurHandler:mil.devcom_sc.ansur.messages')
                runtimeOnly project(':features:equipment::mil.devcom_dac.equipment.handler')
                runtimeOnly project(':features:equipment:mil.devcom_dac.equipment.api')
                runtimeOnly project(':features:equipment:mil.devcom_dac.equipment.handler')
                runtimeOnly project(':features:simplePhysiologyAgent::mil.sstaf.physiology.api')
                runtimeOnly project(':features:simplePhysiologyAgent:mil.sstaf.physiology.agent')
                runtimeOnly project(':features:simplePhysiologyAgent:mil.sstaf.physiology.api')
                runtimeOnly project(':features:simplePhysiologyAgent:mil.sstaf.physiology.models.
api')
                runtimeOnly project(':features:simplePhysiologyAgent:mil.sstaf.physiology.models.
cardiovascular')
                runtimeOnly project(':features:simplePhysiologyAgent:mil.sstaf.physiology.models.
cognition')
                runtimeOnly project(':features:simplePhysiologyAgent:mil.sstaf.physiology.models.
energy')
                runtimeOnly project(':features:simplePhysiologyAgent:mil.sstaf.physiology.models.
hydration')
                runtimeOnly project(':features:simplePhysiologyAgent:mil.sstaf.physiology.models.
musculature')
                runtimeOnly project(':features:simplePhysiologyAgent:mil.sstaf.physiology.models.
respiration')
                runtimeOnly project(':features:simplePhysiologyAgent:mil.sstaf.physiology.models.
vision')
                runtimeOnly project(':features:support:mil.sstaf.blackboard.api')
                runtimeOnly project(':features:support:mil.sstaf.blackboard.inmem')
                runtimeOnly project(':features:support:mil.sstaf.telemetry')
            }
            targets {
                all {
                    testTask.configure {
                        shouldRunAfter(test)
                    }
                }
            }
        }
    }
}
```

A.160 SSTAF/src/framework/mil.sstaf.analyzer/src/main/java/mil/sstaf/analyzer/Analyzer.java

```
package mil.sstaf.analyzer;
import mil.sstaf.analyzer.messages.*;
import mil.sstaf.core.util.SSTAFException;
import mil.sstaf.session.control.Session;
import mil.sstaf.session.messages.BaseSessionCommand;
import mil.sstaf.session.messages.SessionTickResult;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.util.List;
import java.util.Map;
import java.util.Objects;
import java.util.concurrent.*;
import java.util.concurrent.atomic.AtomicBoolean;
import java.util.concurrent.atomic.AtomicLong;
import java.util.function.Consumer;
import java.util.function.Function;
import java.util.function.Supplier;
/**
 * Main processing loop for the service implementations.
 */
public class Analyzer {
    private static final Logger logger = LoggerFactory.getLogger(Analyzer.class);
    private final Session session;
    private final Supplier<String> inputSupplier;
    private final Function<String, BaseAnalyzerCommand> deserializer;
    private final Function<BaseAnalyzerResult, String> serializer;
    private final Consumer<String> outputConsumer;
    private final AtomicBoolean keepRunning = new AtomicBoolean(true);
    private final AtomicBoolean isRunning = new AtomicBoolean(false);
    private final AtomicLong taskCounter = new AtomicLong(0);
    private final ExecutorService executor = Executors.newSingleThreadExecutor();
    /**
     * Evaluation loop task.
     */
    private final Callable<Integer> analysisREPL = new Callable<>() {
        @Override
        public Integer call() {
            //
            // Use anonymous inner classes so that Session already exists when
            // the strategies are instantiated.
            //
            ProcessingStrategy commandListProcessingStrategy = command -> {
                CommandList cl = (CommandList) command;
                for (BaseSessionCommand cmd : cl.getCommands()) {
                    session.submit(cmd);
                }
                SessionTickResult tickResult;
                switch (cl.getMode()) {
                    case SUBMIT_AND_DISPATCH:
                        tickResult = session.tickAgain();
                        break;
                    case TICK:
                        tickResult = session.tick(cl.getTime_ms());
                    case SUBMIT_ONLY:
                    default:
                        tickResult = session.getEmptyTickResult();
                        break;
                }
                return mil.sstaf.analyzer.messages.TickResult.builder()
                        .sessionTickResult(tickResult)
```

```
                    .build();
            };
            ProcessingStrategy tickProcessingStrategy = command -> {
                Tick tick = (Tick) command;
                SessionTickResult tickResult;
                tickResult = session.tick(tick.getTime_ms());
                return mil.sstaf.analyzer.messages.TickResult.builder()
                        .sessionTickResult(tickResult)
                        .build();
            };
            ProcessingStrategy getEntitiesProcessingStrategy = command -> {
                GetEntities getEntities = (GetEntities) command;
                List<String> entityList = session.getEntityController().getEntityPaths();
                return GetEntitiesResult.builder().entities(entityList).build();
            };
            ProcessingStrategy exitStrategy = command -> {
                keepRunning.set(false);
                return null;
            };
            Map<Class<? extends BaseAnalyzerCommand>, ProcessingStrategy> processingStrategyMap =
                    Map.of(Tick.class, tickProcessingStrategy,
                            GetEntities.class, null,
                            CommandList.class, commandListProcessingStrategy);

            isRunning.set(true);
            int rv = 0;
            while (keepRunning.get()) {
                try {
                    logger.debug("Getting command");
                    long startTime = System.currentTimeMillis();
                    String input = inputSupplier.get();
                    logger.debug("Got {}", input);
                    if (input != null && input.length() > 0) {
                        BaseAnalyzerCommand command = deserializer.apply(input);
                        ProcessingStrategy strategy = processingStrategyMap.get(command.getClass(
));

                        BaseAnalyzerResult result;
                        if (strategy == null) {
                            throw new SSTAFException("Unknown Analyzer Command");
                        } else {
                            result = strategy.process(command);
                        }
                        taskCounter.incrementAndGet();
                        result.setId(command.getId());
                        result.setProcessingTime_ms(System.currentTimeMillis() - startTime);
                        String output = serializer.apply(result);
                        outputConsumer.accept(output);
                        logger.info("Sending {}", output);
                    } else if (input != null) {
                        logger.info("Received 0-length command");
                    } else {
                        logger.info("Received null command");
                    }
                } catch (Exception e) {
                    e.printStackTrace();
                    rv = -1;
                    break;
                }
            }
            logger.info("Exiting Analyzer loop");
            isRunning.set(false);
            return rv;
        }
    };

    /**
     * Constructor
     *
```

```
     * @param session        The SSTAF session
     * @param inputSupplier  A {@code Supplier<String>} that provides a {@code String}
     *                       that can be transformed into messages for the
     *                       {@code Session}.
     * @param deserializer   Converts the input String into a {@code BaseAnalyzerCommand}
     * @param serializer     Converts the result into a String
     * @param outputConsumer A {@code Consumer<String>} that accepts the String
     */
    Analyzer(final Session session,
             final Supplier<String> inputSupplier,
             final Function<String, BaseAnalyzerCommand> deserializer,
             final Function<BaseAnalyzerResult, String> serializer,
             final Consumer<String> outputConsumer) {
        this.session = Objects.requireNonNull(session, "session");
        this.inputSupplier = Objects.requireNonNull(inputSupplier, "inputSupplier");
        this.deserializer = Objects.requireNonNull(deserializer, "deserializer");
        this.serializer = Objects.requireNonNull(serializer, "serializer");
        this.outputConsumer = Objects.requireNonNull(outputConsumer, "outputConsumer");
    }
    /**
     * Constructs an {@code Analyzer} using {@code System.in} and
     * {@code System.out}
     *
     * @param session the {@code Session}
     * @return a newly-constructed Analyzer
     */
    public static Analyzer fromSystemIO(final Session session) {
        var reader = new InputStreamReader(System.in);
        var bufferedReader = new BufferedReader(reader);
        var inputSupplier = new MessageReader(bufferedReader);
        var deserializer = new JsonDeserializer();
        var serializer = new JsonSerializer();
        var outputConsumer = new PrintStreamConsumer(System.out);
        return new Analyzer(session, inputSupplier, deserializer,
                serializer, outputConsumer);
    }
    /**
     * Starts the evaluation loop
     */
    public int start() throws InterruptedException, ExecutionException {
        keepRunning.set(true);
        List<Future<Integer>> futures = executor.invokeAll(List.of(analysisREPL));
        Future<Integer> one = futures.get(0);
        return one.get();
    }
    /**
     * Stops the evaluation loop
     */
    public void stop() {
        keepRunning.set(false);
    }
    /**
     * Answers whether the Analyzer is running.
     *
     * @return true if the main loop is running.
     */
    public boolean isRunning() {
        return isRunning.get();
    }
    /**
     * Provides the number of tasks that have been executed by the
     * {@code Analyzer}.
     *
     * @return the number of tasks.
     */
    public long getTaskCount() {
        return taskCounter.get();
    }
```

```
    private interface ProcessingStrategy {
        BaseAnalyzerResult process(BaseAnalyzerCommand command);
    }
}
```

A.161 SSTAF/src/framework/mil.sstaf.analyzer/src/main/java/mil/sstaf/analyzer/FileConsumer.java

```
package mil.sstaf.analyzer;
import mil.sstaf.core.util.SSTAFException;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Objects;
import java.util.function.Consumer;
/**
 * Implements {@code Consumer} to write a {@code String} to the output file.
 */
public class FileConsumer implements Consumer<String> {
    private final BufferedWriter outputWriter;
    public FileConsumer(String fileName) throws IOException {
        Objects.requireNonNull(fileName, "fileName");
        FileWriter fw = new FileWriter(fileName);
        this.outputWriter = new BufferedWriter(fw);
    }
    @Override
    public void accept(String s) {
        try {
            outputWriter.write(s);
            outputWriter.flush();
        } catch (IOException e) {
            throw new SSTAFException("Error writing message", e);
        }
    }
}
```

A.162 SSTAF/src/framework/mil.sstaf.analyzer/src/main/java/mil/sstaf/analyzer/JsonDeserializer.java

```
package mil.sstaf.analyzer;
import mil.sstaf.analyzer.messages.BaseAnalyzerCommand;
import mil.sstaf.core.json.JsonLoader;
import mil.sstaf.core.util.SSTAFException;
import java.nio.file.Path;
import java.util.function.Function;
/**
 * Deserializes a JSON strings into a {@code SessionTasks}
 *
 */
public class JsonDeserializer implements Function<String, BaseAnalyzerCommand> {
    private final JsonLoader jsonLoader = new JsonLoader();
    private final Path fakeFile;
    /**
     * Constructor
     */
    public JsonDeserializer() {
        String cwd = System.getProperty("user.dir");
        fakeFile =Path.of(cwd, "fakeFile.json");
    }
    /**
     * Deserializes the {@code String} into a {@code SessionTas}.
     *
     * A {@code SSTAFException} is thrown if the JSON can not be read into
     * a {@code BaseAnalyzerCommand}
     *
     * @param jsonString the JSON string to be deserialized.
     * @return A SessionTask.
     */
    @Override
    public BaseAnalyzerCommand apply(String jsonString) {
        try {
            return jsonLoader.load(jsonString, BaseAnalyzerCommand.class, fakeFile);
        } catch (Exception e) {
            throw new SSTAFException("Could not convert '" + jsonString
                    + "' into a BaseAnalyzerCommand");
        }
    }
}
```

A.163 SSTAF/src/framework/mil.sstaf.analyzer/src/main/java/mil/sstaf/analyzer/JsonSerializer.java

```
package mil.sstaf.analyzer;
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import mil.sstaf.analyzer.messages.BaseAnalyzerCommand;
import mil.sstaf.analyzer.messages.BaseAnalyzerResult;
import mil.sstaf.core.util.SSTAFException;
import java.util.Objects;
import java.util.function.Function;
public class JsonSerializer implements Function<BaseAnalyzerResult, String> {
    ObjectMapper objectMapper;
    public JsonSerializer() {
        objectMapper = new ObjectMapper();
    }
    @Override
    public String apply(BaseAnalyzerResult o) {
        Objects.requireNonNull(o);
        try {
            return objectMapper.writeValueAsString(o);
        } catch (JsonProcessingException e) {
            throw new SSTAFException("Could not serialize " + o.getClass().getName(), e);
        }
    }
}
```

A.164 SSTAF/src/framework/mil.sstaf.analyzer/src/main/java/mil/sstaf/analyzer/Main.java

```
package mil.sstaf.analyzer;
import mil.sstaf.core.configuration.SSTAFConfiguration;
import mil.sstaf.session.control.Session;
import mil.sstaf.session.control.SessionConfiguration;
import mil.sstaf.session.control.EntityController;
import org.slf4j.LoggerFactory;
import org.slf4j.Logger;
import java.io.File;
import java.util.concurrent.ExecutionException;
import static java.lang.System.exit;
/**
 * The entry point for the Analyzer
 */
public class Main {
    public static final int ERROR_NO_ARGS = 1;
    public static final int ERROR_ENTITY_FILE_NOT_READABLE = 2;
    private static final Logger logger = LoggerFactory.getLogger(Main.class);
    /**
     * The Main method
     */
    public static void main(String[] args) {
        if (args.length != 1) {
            printUsage();
            exit(ERROR_NO_ARGS);
        }
        String entityFileName = args[0];
        File entityFile = new File(entityFileName);
        if (!entityFile.exists() | !entityFile.canRead()) {
            System.err.printf("Entity file '%s' does not exist or is not readable\n", entityFileN
ame);
            exit(ERROR_ENTITY_FILE_NOT_READABLE);
        }
        EntityController entityController = EntityController.from(entityFile);
        SSTAFConfiguration config = SSTAFConfiguration.getInstance();
        SessionConfiguration sessionConfiguration;
        if (config instanceof SessionConfiguration) {
            sessionConfiguration = (SessionConfiguration) config;
        } else {
            sessionConfiguration = SessionConfiguration.builder().build();
            logger.info("Creating default configuration");
        }
        Session session = Session.of(sessionConfiguration, entityController);
        Analyzer analyzer = Analyzer.fromSystemIO(session);
        logger.info("Starting analyzer loop");
        int rv;
        try {
            rv = analyzer.start();
        } catch (InterruptedException | ExecutionException e) {
            throw new RuntimeException(e);
        }
        System.exit(rv);
    }
    /**
     * Print Usage.
     */
    public static void printUsage() {
        System.err.println("Usage: sstaf-analyzer [-Dmil.sstaf.configuration=configFile]  entityF
ile");
    }
}
```

A.165 SSTAF/src/framework/mil.sstaf.analyzer/src/main/java/mil/sstaf/analyzer/MessageReader.java

```
package mil.sstaf.analyzer;
import lombok.Getter;
import mil.sstaf.core.util.SSTAFException;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import java.io.BufferedReader;
import java.io.IOException;
import java.util.Objects;
import java.util.function.Supplier;
public class MessageReader implements Supplier<String> {
    @Getter
    private final BufferedReader reader;
    @Getter
    private int openCurlies = 0;
    @Getter
    private int openSquares = 0;
    private static final Logger logger = LoggerFactory.getLogger(MessageReader.class);
    public MessageReader(BufferedReader reader) {
        this.reader = Objects.requireNonNull(reader, "reader");
    }
    String scanAndCleanLine(String line) {
        String clean = line.replaceAll("\\R", " ");
        for (char c : clean.toCharArray()) {
            if (c == '{') ++openCurlies;
            else if (c == '}') --openCurlies;
            else if (c == '[') ++openSquares;
            else if (c == ']') --openSquares;
        }
        if (openCurlies < 0) {
            throw new SSTAFException("Encountered extra '}' in '" + clean + "'");
        }
        if (openSquares < 0) {
            throw new SSTAFException("Encountered extra ']' in '" + clean + "'");
        }
        return clean;
    }
    public String get() {
        logger.info("getting...");
        StringBuilder sb = new StringBuilder();
        try {
            while (true) {
                String line = reader.readLine();
                logger.info("Read {}", line);
                if (line == null) {
                    break;
                } else {
                    String clean = scanAndCleanLine(line);
                    sb.append(clean);
                    if (openCurlies == 0 && openSquares == 0) {
                        break;
                    }
                }
            }
            if (openCurlies != 0 || openSquares != 0) {
                throw new SSTAFException("Malformed JSON command: '" + sb + "'");
            }
            return sb.toString();
        } catch (IOException ioe) {
            throw new SSTAFException("Could not read message",ioe);
        }
    }
}
```

A.166 SSTAF/src/framework/mil.sstaf.analyzer/src/main/java/mil/sstaf/analyzer/PrintStreamConsumer.java

```
package mil.sstaf.analyzer;
import mil.sstaf.core.util.SSTAFException;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import java.io.PrintStream;
import java.util.Objects;
import java.util.function.Consumer;
/**
 * Implements {@code Consumer} to write a {@code String} to the output.
 */
public class PrintStreamConsumer implements Consumer<String> {
    private static final Logger logger = LoggerFactory.getLogger(PrintStreamConsumer.class);
    private final PrintStream outputWriter;
    public PrintStreamConsumer(PrintStream outputWriter) {
        this.outputWriter = Objects.requireNonNull(outputWriter, "outputWriter");
    }
    @Override
    public void accept(String s) {
        String toSend = s + "\n";
        logger.info("Writing {}", toSend);
        outputWriter.println(toSend);
        outputWriter.flush();
        if (outputWriter.checkError()) {
            throw new SSTAFException("Error writing message");
        }
    }
}
```

A.167 SSTAF/src/framework/mil.sstaf.analyzer/src/main/java/mil/sstaf/analyzer/http/HttpService.java

```
package mil.sstaf.analyzer.http;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
public class HttpService {
    private static final Logger logger = LoggerFactory.getLogger(HttpService.class);
    public static void main(String[] args) {
        logger.error("This service is not implemented. I fear STIGs");
    }
}
```

A.168 SSTAF/src/framework/mil.sstaf.analyzer/src/main/java/mil/sstaf/analyzer/messages/BaseAnalyzerCommand.java

```
package mil.sstaf.analyzer.messages;
import com.fasterxml.jackson.annotation.JsonIgnore;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.Builder;
import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import java.util.concurrent.atomic.AtomicLong;
@Jacksonized
@SuperBuilder
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
@EqualsAndHashCode
public class BaseAnalyzerCommand {
    private static final AtomicLong counter = new AtomicLong(0);
    @Getter
    @Builder.Default
    @JsonIgnore
    private long id = counter.getAndIncrement();
}
```

A.169 SSTAF/src/framework/mil.sstaf.analyzer/src/main/java/mil/sstaf/analyzer/messages/BaseAnalyzerResult.java

```
package mil.sstaf.analyzer.messages;
import com.fasterxml.jackson.annotation.JsonIgnore;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.Builder;
import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.Setter;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
@Jacksonized
@SuperBuilder
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
@EqualsAndHashCode
public class BaseAnalyzerResult {
    @Getter
    @Setter
    private long id;
    @Getter
    @Setter
    private long processingTime_ms;
}
```

A.170 SSTAF/src/framework/mil.sstaf.analyzer/src/main/java/mil/sstaf/analyzer/messages/CommandList.java

```
package mil.sstaf.analyzer.messages;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.NonNull;
import lombok.Singular;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import mil.sstaf.session.messages.BaseSessionCommand;
import java.util.List;
@Jacksonized
@SuperBuilder
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
@EqualsAndHashCode(callSuper = true)
public class CommandList extends BaseAnalyzerCommand {
    @Getter
    @NonNull
    @Singular
    private List<BaseSessionCommand> commands;
    @Getter
    @NonNull
    private Mode mode;
    @Getter
    private long time_ms;
}
```

A.171 SSTAF/src/framework/mil.sstaf.analyzer/src/main/java/mil/sstaf/analyzer/messages/Exit.java

```
package mil.sstaf.analyzer.messages;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.EqualsAndHashCode;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
@Jacksonized
@SuperBuilder
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
@EqualsAndHashCode(callSuper = true)
public class Exit extends BaseAnalyzerCommand {
}
```

A.172 SSTAF/src/framework/mil.sstaf.analyzer/src/main/java/mil/sstaf/analyzer/messages/GetEntities.java

```
package mil.sstaf.analyzer.messages;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.EqualsAndHashCode;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
@Jacksonized
@SuperBuilder
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
@EqualsAndHashCode(callSuper = true)
public class GetEntities extends BaseAnalyzerCommand {
}
```

A.173 SSTAF/src/framework/mil.sstaf.analyzer/src/main/java/mil/sstaf/analyzer/messages/GetEntitiesResult.java

```
package mil.sstaf.analyzer.messages;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.Singular;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import java.util.List;
@Jacksonized
@SuperBuilder
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
@EqualsAndHashCode(callSuper = true)
public class GetEntitiesResult extends BaseAnalyzerResult {
    @Getter
    @Singular
    List<String> entities;
}
```

A.174 SSTAF/src/framework/mil.sstaf.analyzer/src/main/java/mil/sstaf/analyzer/messages/Mode.java

```
package mil.sstaf.analyzer.messages;
public enum Mode {
    /**
     * Submits the commands to the {@code EntityController} but does
     * not dispatch them.
     */
    SUBMIT_ONLY,
    /**
     * Submits the commands to the {@code EntityController} and dispatches
     * them to the {@code Entities} by executing a tick with a zero delta-t.
     */
    SUBMIT_AND_DISPATCH,
    /**
     * Submits the commands and then executes a tick using the new simulation
     * clock time provided in the {@code CommandList}.
     */
    TICK
}
```

A.175 SSTAF/src/framework/mil.sstaf.analyzer/src/main/java/mil/sstaf/analyzer/messages/Tick.java

```
package mil.sstaf.analyzer.messages;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
@Jacksonized
@SuperBuilder
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
@EqualsAndHashCode(callSuper = true)
public class Tick extends BaseAnalyzerCommand {
    @Getter
    private long time_ms;
}
```

A.176 SSTAF/src/framework/mil.sstaf.analyzer/src/main/java/mil/sstaf/analyzer/messages/TickResult.java

```
package mil.sstaf.analyzer.messages;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.NonNull;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import mil.sstaf.session.messages.SessionTickResult;
@Jacksonized
@SuperBuilder
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
@EqualsAndHashCode(callSuper = true)
public class TickResult extends BaseAnalyzerResult {
    @NonNull
    @Getter
    SessionTickResult sessionTickResult;
}
```

A.177 SSTAF/src/framework/mil.sstaf.analyzer/src/main/java/mil/sstaf/analyzer/pipe/PipeService.java

```
package mil.sstaf.analyzer.pipe;
import java.io.*;
import java.util.function.Consumer;
import java.util.function.Supplier;
public class PipeService {
    public static void printUsage() {
        String message = "Usage: java -p SSTAF_MODULE_DIR -m mil.sstaf.service/mil.sstaf.service.
pipe.Main sessionConfigFile.json";
        System.err.println(message);
    }
    private final static Supplier<String> supplier = new Supplier<>() {
        final BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
        @Override
        public String get() {
            String rv = "";
            try {
                rv = in.readLine();
            } catch (IOException e) {
                e.printStackTrace();
            }
            return rv;
        }
    };
    private final static Consumer<String> consumer = new Consumer<>() {
        final BufferedWriter out = new BufferedWriter(new OutputStreamWriter(System.out));
        @Override
        public void accept(String s) {
            try {
                out.write(s);
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    };
    public static void main(String[] args) {
        if (args.length < 1) {
            printUsage();
            System.exit(-1);
        }
        String configFile = args[0];
       // Session session = Session.factory(configFile).build();

    }
}
```

A.178 SSTAF/src/framework/mil.sstaf.analyzer/src/main/java/module-info.java

```
module mil.sstaf.analyzer {
    exports mil.sstaf.analyzer.messages;
    requires mil.sstaf.core;
    requires mil.sstaf.session;
    requires org.slf4j;
    opens mil.sstaf.analyzer.messages to com.fasterxml.jackson.databind;
}
```

A.179 SSTAF/src/framework/mil.sstaf.analyzer/src/test/java/mil/sstaf/analyzer/AnalyzerTest.java

```
package mil.sstaf.analyzer;
import mil.sstaf.analyzer.messages.BaseAnalyzerCommand;
import mil.sstaf.analyzer.messages.Tick;
import mil.sstaf.session.control.Session;
import mil.sstaf.session.control.SessionConfiguration;
import mil.sstaf.session.control.EntityController;
import org.junit.jupiter.api.*;
import java.io.File;
import java.util.ArrayDeque;
import java.util.ArrayList;
import java.util.Deque;
import java.util.List;
import java.util.concurrent.ExecutionException;
import java.util.function.Consumer;
import java.util.function.Supplier;
import static java.lang.Thread.sleep;
import static org.junit.jupiter.api.Assertions.*;

@Disabled
public class AnalyzerTest {
    static final String GET_ENTITIES_MSG = "{ \"class\" : \"mil.sstaf.analyzer.commands.GetEntiti
esQuery\" }\n";
    public static final String END_SESSION_MSG = "{ \"class\" : \"mil.sstaf.analyzer.commands.End
Session\" }\n";
    private String mockStringSupplier() {
        return "";
    }
    private BaseAnalyzerCommand mockDeserializer(String s) {
        return Tick.builder().time_ms(0).build();
    }
    private void mockStringConsumer(String out) {
    }
    private String mockSerializer(Object o) {
        return "";
    }
    static class StringSupplier implements Supplier<String> {
        Deque<String> queue = new ArrayDeque<>(20);
        void add(String s) {
            queue.offer(s);
        }
        @Override
        public String get() {
            return queue.removeFirst();
        }
    }

    static class StringConsumer implements Consumer<String> {
        StringBuilder stringBuilder = new StringBuilder();
        List<String> stuff = new ArrayList<>();
        @Override
        public void accept(String s) {
            stringBuilder.append(s);
            stringBuilder.append("\n");
            stuff.add(stringBuilder.toString());
        }
    }
    @Nested
    @DisplayName("Test normal behavior (The Happy Path)")
    class HappyTests {
        @Test
        @DisplayName("Confirm an Analyzer can be created")
        void testCreation() {
```

```
            File entityFile = new File("src/test/resources/analyzerTest/entityConfig.json");
            EntityController entityController = EntityController.from(entityFile);
            SessionConfiguration configuration = SessionConfiguration
                    .builder()
                    .async(false)
                    .build();
            Session session = Session.of(configuration, entityController);
            assertDoesNotThrow(() -> new Analyzer(
                    session,
                    AnalyzerTest.this::mockStringSupplier,
                    AnalyzerTest.this::mockDeserializer,
                    AnalyzerTest.this::mockSerializer,
                    AnalyzerTest.this::mockStringConsumer));
        }
        @Test
        @DisplayName("EndSession works")
        void testEndSession() throws InterruptedException, ExecutionException {
            StringSupplier sup = new StringSupplier();
            StringConsumer con = new StringConsumer();
            sup.add(END_SESSION_MSG);
            Analyzer analyzer = makeAnalyzer(sup, con);
            assertEquals(0, analyzer.getTaskCount());
            assertFalse(analyzer.isRunning());
            analyzer.start();
            sleep(1000);
            assertFalse(analyzer.isRunning());
            assertEquals(1, analyzer.getTaskCount());
            assertTrue(sup.queue.isEmpty());
        }
        @Test
        @DisplayName("GetEntitiesQuery works")
        void testGetEntities() throws InterruptedException, ExecutionException {
            StringSupplier sup = new StringSupplier();
            StringConsumer con = new StringConsumer();
            sup.add(GET_ENTITIES_MSG);
            sup.add(END_SESSION_MSG);;
            Analyzer analyzer = makeAnalyzer(sup, con);
            assertEquals(0, analyzer.getTaskCount());
            assertFalse(analyzer.isRunning());
            analyzer.start();
            sleep(1000);
            assertEquals(2, con.stuff.size());
            assertFalse(analyzer.isRunning());
            assertEquals(2, analyzer.getTaskCount());
            assertTrue(sup.queue.isEmpty());
        }

        @Test
        @DisplayName("Tick works")
        void testTickSession() throws InterruptedException, ExecutionException {
            StringSupplier sup = new StringSupplier();
            StringConsumer con = new StringConsumer();
            String tick = "{ \"class\" : \"mil.sstaf.analyzer.commands.Tick\", \"time_ms\" : 1000
0 }\n";
            sup.add(tick);
            sup.add(END_SESSION_MSG);
            Analyzer analyzer = makeAnalyzer(sup, con);
            assertEquals(0, analyzer.getTaskCount());
            assertFalse(analyzer.isRunning());
            analyzer.start();
            sleep(1000);
            assertFalse(analyzer.isRunning());
            assertEquals(2, analyzer.getTaskCount());
            assertTrue(sup.queue.isEmpty());
        }

        private Analyzer makeAnalyzer(Supplier<String> in, Consumer<String> out) {
            File entityFile = new File("src/test/resources/analyzerTest/entityConfig.json");
```

```
            Analyzer analyzer = getAnalyzer(in, out, entityFile);
            return analyzer;
        }
        private Analyzer makePlatoonAnalyzer(Supplier<String> in, Consumer<String> out) {
            File entityFile = new File("../../testInput/goodEntityFiles/OneSoldier.json");
            Analyzer analyzer = getAnalyzer(in, out, entityFile);
            return analyzer;
        }
    }
    private Analyzer getAnalyzer(Supplier<String> in, Consumer<String> out, File entityFile) {
        EntityController entityController = EntityController.from(entityFile);
        SessionConfiguration configuration = SessionConfiguration
                .builder()
                .async(false)
                .build();
        Session session = Session.of(configuration, entityController);
        Analyzer analyzer = new Analyzer(session,
                in,
                new JsonDeserializer(),
                new JsonSerializer(),
                out);
        assertNotNull(analyzer);
        return analyzer;
    }
    @Nested
    @DisplayName("Test failure behavior (The Unhappy Paths")
    class UnhappyTests {
        @Test
        @DisplayName("Confirm null session throws")
        void testNullSession() {
            assertThrows(NullPointerException.class, () -> new Analyzer(null,
                    AnalyzerTest.this::mockStringSupplier,
                    AnalyzerTest.this::mockDeserializer,
                    AnalyzerTest.this::mockSerializer,
                    AnalyzerTest.this::mockStringConsumer));
        }
        @Test
        @DisplayName("Confirm null string source fails throws")
        void testNullSource() {
            assertThrows(NullPointerException.class, () -> {
                File entityFile = new File("src/test/resources/analyzerTest/entityConfig.json");
                EntityController entityController = EntityController.from(entityFile);
                SessionConfiguration configuration = SessionConfiguration
                        .builder()
                        .async(false)
                        .build();
                try (Session session = Session.of(configuration, entityController)) {
                    Analyzer analyzer = new Analyzer(session,
                            null,
                            AnalyzerTest.this::mockDeserializer,
                            AnalyzerTest.this::mockSerializer,
                            AnalyzerTest.this::mockStringConsumer
                    );
                    assertNotNull(analyzer);
                }
            });
        }
        @Test
        @DisplayName("Confirm that a null converter throws")
        void testNullConverter() {
            assertThrows(NullPointerException.class, () -> {
                File entityFile = new File("src/test/resources/analyzerTest/entityConfig.json");
                EntityController entityController = EntityController.from(entityFile);
                SessionConfiguration configuration = SessionConfiguration
                        .builder()
                        .async(false)
                        .build();
                try (Session session = Session.of(configuration, entityController)) {
```

```
                    Analyzer analyzer = new Analyzer(session,
                            AnalyzerTest.this::mockStringSupplier,
                            null,
                            AnalyzerTest.this::mockSerializer,
                            AnalyzerTest.this::mockStringConsumer
                    );
                    assertNotNull(analyzer);
                }
            });
        }
        @Test
        @DisplayName("Confirm that a null serializer throws")
        void testNullSerializer() {
            assertThrows(NullPointerException.class, () -> {
                File entityFile = new File("src/test/resources/analyzerTest/entityConfig.json");
                EntityController entityController = EntityController.from(entityFile);
                SessionConfiguration configuration = SessionConfiguration
                        .builder()
                        .async(false)
                        .build();
                try (Session session = Session.of(configuration, entityController)) {
                    Analyzer analyzer = new Analyzer(session,
                            AnalyzerTest.this::mockStringSupplier,
                            AnalyzerTest.this::mockDeserializer,
                            null,
                            AnalyzerTest.this::mockStringConsumer
                    );
                    assertNotNull(analyzer);
                }
            });
        }
        @Test
        @DisplayName("Confirm that a null String consumer throws")
        void testNullDestination() {
            assertThrows(NullPointerException.class, () -> {
                File entityFile = new File("src/test/resources/analyzerTest/entityConfig.json");
                EntityController entityController = EntityController.from(entityFile);
                SessionConfiguration configuration = SessionConfiguration
                        .builder()
                        .async(false)
                        .build();
                try (Session session = Session.of(configuration, entityController)) {
                    Analyzer analyzer = new Analyzer(session,
                            AnalyzerTest.this::mockStringSupplier,
                            AnalyzerTest.this::mockDeserializer,
                            AnalyzerTest.this::mockSerializer,
                            null
                    );
                    assertNotNull(analyzer);
                }
            });
        }
    }
}
```

A.180 SSTAF/src/framework/mil.sstaf.analyzer/src/test/java/mil/sstaf/analyzer/MessageConverterTest.java

```
package mil.sstaf.analyzer;
import mil.sstaf.analyzer.messages.BaseAnalyzerCommand;
import mil.sstaf.analyzer.messages.Tick;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Nested;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
public class MessageConverterTest {
    private JsonDeserializer messageConverter;
    @BeforeEach
    void setup() {
        messageConverter = new JsonDeserializer();
    }
    @Nested
    @DisplayName("Test normal behavior (The Happy Path)")
    class HappyTests {
        @Test
        @DisplayName("Confirm a Tick can be generated")
        void tickTest() {
            assertDoesNotThrow(() -> {
                String json = "{ \"class\" : \"mil.sstaf.analyzer.messages.Tick\","
                        + "\"time_ms\" : 1000 }";
                BaseAnalyzerCommand thing = messageConverter.apply(json);
                assertNotNull(thing);
                assertTrue(thing instanceof Tick);
            });
        }

    }
}
```

A.181 SSTAF/src/framework/mil.sstaf.analyzer/src/test/java/mil/sstaf/analyzer/MessageReaderTest.java

```
package mil.sstaf.analyzer;
import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.node.ObjectNode;
import mil.sstaf.core.util.SSTAFException;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Nested;
import org.junit.jupiter.api.Test;
import java.io.BufferedReader;
import java.io.StringReader;
import static org.junit.jupiter.api.Assertions.*;
/**
 * Tests the MessageReader class
 */
public class MessageReaderTest {
    private MessageReader makeReader(String json) {
        StringReader sr = new StringReader(json);
        BufferedReader br = new BufferedReader(sr);
        return new MessageReader(br);
    }
    @Nested
    @DisplayName("Test scanAndCleanLine scenarios")
    class ScanTests {
        @Test
        @DisplayName("Check single open")
        void testSingle() {
            assertDoesNotThrow(() -> {
                String json = "{";
                MessageReader messageReader = makeReader(json);
                assertNotNull(messageReader);
                String valid = messageReader.scanAndCleanLine(json);
                assertEquals(1, messageReader.getOpenCurlies());
                assertEquals(0, messageReader.getOpenSquares());
                assertEquals(json, valid);
            });
        }
        @Test
        @DisplayName("Check open and closed")
        void testOpenAndClosed() {
            assertDoesNotThrow(() -> {
                String json = "{[]}";
                MessageReader messageReader = makeReader(json);
                assertNotNull(messageReader);
                String valid = messageReader.scanAndCleanLine(json);
                assertEquals(0, messageReader.getOpenCurlies());
                assertEquals(0, messageReader.getOpenSquares());
                assertEquals(json, valid);
            });
        }
        @Test
        @DisplayName("A multi-line Linux string gets flattened")
        void testFlattening1() {
            assertDoesNotThrow(() -> {
                String json = "{\n[\n]\n}";
                MessageReader messageReader = makeReader(json);
                assertNotNull(messageReader);
                String valid = messageReader.scanAndCleanLine(json);
                assertEquals(0, messageReader.getOpenCurlies());
                assertEquals(0, messageReader.getOpenSquares());
                assertEquals("{ [ ] }", valid);
            });
        }
```

```
        @Test
        @DisplayName("A multi-line MacOS string gets flattened")
        void testFlattening2() {
            assertDoesNotThrow(() -> {
                String json = "{\r[\r]\r}";
                MessageReader messageReader = makeReader(json);
                assertNotNull(messageReader);
                String valid = messageReader.scanAndCleanLine(json);
                assertEquals(0, messageReader.getOpenCurlies());
                assertEquals(0, messageReader.getOpenSquares());
                assertEquals("{ [ ] }", valid);
            });
        }
        @Test
        @DisplayName("A multi-line Windows string gets flattened")
        void testFlattening3() {
            assertDoesNotThrow(() -> {
                String json = "{\r\n[\r\n]\r\n}";
                MessageReader messageReader = makeReader(json);
                assertNotNull(messageReader);
                String valid = messageReader.scanAndCleanLine(json);
                assertEquals(0, messageReader.getOpenCurlies());
                assertEquals(0, messageReader.getOpenSquares());
                assertEquals("{ [ ] }", valid);
            });
        }
        @Test
        @DisplayName("Extra '}' throws")
        void extraCloseCurlyThrows() {
            SSTAFException sstafException =
                    assertThrows(SSTAFException.class, () -> {
                        String json = "{\r\n[\r\n]\r\n}}";
                        MessageReader messageReader = makeReader(json);
                        assertNotNull(messageReader);
                        messageReader.scanAndCleanLine(json);
                    });
            assertTrue(sstafException.getMessage().contains("Encountered extra '}'"));
        }
        @Test
        @DisplayName("Extra ']' throws")
        void extraCloseSquareThrows() {
            SSTAFException sstafException =
                    assertThrows(SSTAFException.class, () -> {
                        String json = "{\r\n[\r]\n]\r\n}}";
                        MessageReader messageReader = makeReader(json);
                        assertNotNull(messageReader);
                        messageReader.scanAndCleanLine(json);
                    });
            assertTrue(sstafException.getMessage().contains("Encountered extra '}'"));
        }
    }
    @Nested
    @DisplayName("Process well-formed messages (The Happy Path)")
    class HappyTests {
        @Test
        @DisplayName("Test empty on 1 line")
        void singleLineTest() {
            assertDoesNotThrow(() -> {
                String json = "{}";
                MessageReader messageReader = makeReader(json);
                assertNotNull(messageReader);
                String valid = messageReader.get();
                assertEquals(json, valid);
            });
        }
        @Test
        @DisplayName("Test complicated on multiple lines")
        void testComplicated() {
```

```
            assertDoesNotThrow(() -> {
                String json = "{ \"x\" : \"y\",\n \"p\": \n [ \"a\" , \"b\" ] }";
                MessageReader messageReader = makeReader(json);
                assertNotNull(messageReader);
                String valid = messageReader.get();
                ObjectMapper om = new ObjectMapper();
                JsonNode tree = om.readTree(valid);
                assertTrue(tree.isObject());
                assertTrue(tree.has("x"));
                assertTrue(tree.has("p"));
            });
        }
        @Test
        @DisplayName("A sequence of complicated messages on multiple lines can be read")
        void testVeryComplicated() {
            assertDoesNotThrow(() -> {
                String json = "{ \"x\" : 0,\n \"p\": \n [ \"a\" , \"b\" ] }\n"
                        + "{ \"x\" : 1,\n \"p\": \n [ \"a\" , \"b\" ] }\n"
                        + "{ \"x\" : 2,\n \"p\": \n [ \"a\" , \"b\" ] }\n"
                        + "{ \"x\" : 3,\n \"p\": \n [ \"a\" , \"b\" ] }\n"
                        + "{ \"x\" : 4,\n \"p\": \n [ \"a\" , \"b\" ] }\n";
                MessageReader messageReader = makeReader(json);
                assertNotNull(messageReader);
                for (int i = 0; i < 5; ++i) {
                    String valid = messageReader.get();
                    ObjectMapper om = new ObjectMapper();
                    JsonNode tree = om.readTree(valid);
                    assertTrue(tree.isObject());
                    ObjectNode obj = (ObjectNode) tree;
                    assertTrue(obj.has("x"));
                    assertEquals(i, obj.get("x").asInt());
                    assertTrue(tree.has("p"));
                }
            });
        }
    }
    @Nested
    @DisplayName("Test failure modes (The Unhappy Paths)")
    class UnhappyTests {
        @Test
        @DisplayName("Extra '}' throws")
        void testExtraCloseCurly() {
            SSTAFException sstafException = assertThrows(SSTAFException.class, () -> {
                String json = "}";
                MessageReader messageReader = makeReader(json);
                assertNotNull(messageReader);
                messageReader.get();
            });
            assertTrue(sstafException.getMessage().contains("'}'"));
        }
        @Test
        @DisplayName("Test bad complicated on multiple lines")
        void testComplicated() {
            SSTAFException sstafException = assertThrows(SSTAFException.class, () -> {
                String json = "{ \"x\" : \"y\",\n \"p\": \n [ [ \"a\" , \"b\" ] }";
                MessageReader messageReader = makeReader(json);
                assertNotNull(messageReader);
                messageReader.get();
            });
            assertTrue(sstafException.getMessage().contains("Malformed JSON"));
        }
    }
}
```

A.182 SSTAF/src/framework/mil.sstaf.analyzer/src/test/resources/analyzerTest/SoldierConfiguration/TestSoldier1.json

```
{
  "class" : "mil.sstaf.core.entity.Soldier",
  "name": "Test Dude",
  "rank": "E6",
  "features": [
  ],
  "configurations": {}
}
```

A.183 SSTAF/src/framework/mil.sstaf.analyzer/src/test/resources/analyzerTest/SoldierConfiguration/TestSoldier2.json

```
{
  "class" : "mil.sstaf.core.entity.Soldier",
  "name": "Test Dude 2",
  "rank": "E6",
  "features": [
  ],
  "configurations": {}
}
```

A.184 SSTAF/src/framework/mil.sstaf.analyzer/src/test/resources/analyzerTest/SoldierConfiguration/TestSoldier6.json

```
{
  "class" : "mil.sstaf.core.entity.Soldier",
  "name": "Test Dude",
  "rank": "E4",
  "features": [
  ],
  "configurations": {}
}
```

A.185 SSTAF/src/framework/mil.sstaf.analyzer/src/test/resources/analyzerTest/UnitConfiguration/Company.json

```
{
  "class" : "mil.sstaf.core.entity.Unit",
  "name": "Test Platoon",
  "type": "Platoon",
  "features": [],
  "configurations": {},
  "soldiers": [
    {
      "position": "CC",
      "soldier": "../SoldierConfiguration/TestSoldier1.json"
    },
    {
      "position": "DCC",
      "soldier": "../SoldierConfiguration/TestSoldier1.json"
    },
    {
      "position": "G2",
      "soldier": "../SoldierConfiguration/TestSoldier1.json"
    }
  ],
  "subUnits": [
    {
      "label": "1st Platoon",
      "unit": "Platoon.json"
    },
    {
      "label": "2nd Platoon",
      "unit": "Platoon.json"
    },
    {
      "label": "3rd Platoon",
      "unit": "Platoon.json"
    }
  ]
}
```

A.186 SSTAF/src/framework/mil.sstaf.analyzer/src/test/resources/analyzerTest/UnitConfiguration/FireTeam.json

```
{
  "class" : "mil.sstaf.core.entity.Unit",
  "name": "Test Unit",
  "type": "FireTeam",
  "features": [
  ],
  "configurations": {},
  "soldiers": [
    {
      "position": "TL",
      "soldier": "../SoldierConfiguration/TestSoldier1.json"
    },
    {
      "position": "R",
      "soldier": "../SoldierConfiguration/TestSoldier2.json"
    },
    {
      "position": "G",
      "soldier": "../SoldierConfiguration/TestSoldier1.json"
    },
    {
      "position": "AR",
      "soldier": "../SoldierConfiguration/TestSoldier2.json"
    }
  ]
}
```

A.187 SSTAF/src/framework/mil.sstaf.analyzer/src/test/resources/analyzerTest/UnitConfiguration/Platoon.json

```
{
  "class" : "mil.sstaf.core.entity.Unit",
  "name": "Test Platoon",
  "type": "Platoon",
  "features": [],
  "configurations": {},
  "soldiers": [
    {
      "position": "PL",
      "soldier": "../SoldierConfiguration/TestSoldier1.json"
    },
    {
      "position": "DPL",
      "soldier": "../SoldierConfiguration/TestSoldier1.json"
    }
  ],
  "subUnits": [
    {
      "label": "Squad A",
      "unit": "Squad.json"
    },
    {
      "label": "Squad B",
      "unit": "Squad.json"
    },
    {
      "label": "Squad C",
      "unit": "Squad.json"
    }
  ]
}
```

A.188 SSTAF/src/framework/mil.sstaf.analyzer/src/test/resources/analyzerTest/UnitConfiguration/Squad.json

```
{
  "class" : "mil.sstaf.core.entity.Unit",
  "name": "Test Squad",
  "type": "Squad",
  "features": [
  ],
  "configurations": {},
  "soldiers": [
    {
      "position": "SL",
      "soldier": "../SoldierConfiguration/TestSoldier1.json"
    }
  ],
  "subUnits": [
    {
      "label": "Fire Team Alpha",
      "unit": "FireTeam.json"
    },
    {
      "label": "Fire Team Bravo",
      "unit": "FireTeam.json"
    }
  ]
}
```

A.189 SSTAF/src/framework/mil.sstaf.analyzer/src/test/resources/analyzerTest/entityConfig.json

```
{
  "class": "mil.sstaf.session.control.EntityController",
  "entities": {
    "BLUE": [
      "UnitConfiguration/Platoon.json",
      "UnitConfiguration/Platoon.json",
      "UnitConfiguration/Platoon.json",
      "UnitConfiguration/Platoon.json",
      "UnitConfiguration/Platoon.json"
    ],
    "RED": [
      "UnitConfiguration/Platoon.json",
      "UnitConfiguration/Platoon.json",
      "UnitConfiguration/Platoon.json",
      "UnitConfiguration/Platoon.json",
      "UnitConfiguration/Platoon.json"
    ],
    "GRAY": []
  },
  "features": [],
  "configurations": {},
  "randomSeed": 3
}
```

A.190 SSTAF/src/framework/mil.sstaf.core/build.gradle

```
dependencies {
    implementation group: 'org.apache.commons', name: 'commons-math3', version: '3.6.1'
}
test.dependsOn(':testModules:mil.sstaftest.fred:build')
test.dependsOn(':testModules:mil.sstaftest.barney:build')
test.dependsOn(':testModules:mil.sstaftest.wilma:build')
test.dependsOn(':testModules:mil.sstaftest.betty:build')
integrationTest.dependsOn(':testFeatures:integration:mil.sstaftest.jamesbond:copyJar')
integrationTest.dependsOn(':testFeatures:integration:mil.sstaftest.alpha:copyJar')
integrationTest.dependsOn(':testFeatures:integration:mil.sstaftest.bravo:copyJar')
integrationTest.dependsOn(':testFeatures:integration:mil.sstaftest.charlie:copyJar')
integrationTest.dependsOn(':testFeatures:integration:mil.sstaftest.delta:copyJar')
integrationTest.dependsOn(':testFeatures:integration:mil.sstaftest.echo:copyJar')
integrationTest.dependsOn(':testFeatures:support:mil.sstaftest.mocks.pinky:copyJar')
testing {
    suites {
        integrationTest {
            dependencies {
                implementation project(':framework:mil.sstaf.core')
                implementation project(':testFeatures:integration:mil.sstaftest.simplemock')
                implementation project(':testFeatures:integration:mil.sstaftest.jamesbond')
                implementation project(':testFeatures:integration:mil.sstaftest.alpha')
                implementation project(':testFeatures:integration:mil.sstaftest.bravo')
                implementation project(':testFeatures:integration:mil.sstaftest.charlie')
                implementation project(':testFeatures:integration:mil.sstaftest.delta')
                implementation project(':testFeatures:integration:mil.sstaftest.echo')
                implementation project(':testFeatures:support:mil.sstaftest.mocks.pinky')
                runtimeOnly project(':testFeatures:support:mil.sstaftest.mocks.handler1')
                runtimeOnly project(':testFeatures:support:mil.sstaftest.mocks.agent1')
                runtimeOnly project(':testFeatures:support:mil.sstaftest.mocks.pinky')
            }
        }
    }
}
```

A.191 SSTAF/src/framework/mil.sstaf.core/src/integrationTest/java/mil/sstaf/core/integration/AppSupportTest.java

```
package mil.sstaf.core.integration;
import mil.sstaf.core.features.*;
import mil.sstaf.core.util.SSTAFException;
import mil.sstaftest.simplemock.SimpleMockFeature;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
import java.io.File;
import java.io.IOException;
import java.util.List;
import java.util.Locale;
import static org.junit.jupiter.api.Assertions.assertDoesNotThrow;
public class AppSupportTest {

    @Test
    @DisplayName("Confirm that a transient AppSession can be created to run the 'upper.py' script
")
    void toUpperTestTransient() {
        assertDoesNotThrow(() -> {
            AppConfiguration config = AppConfiguration.builder()
                    .resourceOwner(SimpleMockFeature.class)
                    .resource("mil/sstaftest/simplemock/upper.py")
                    .mode(AppSupport.Mode.TRANSIENT)
                    .build();
            AppAdapter helper = AppSupport.createAdapter(config);
            ResourceManager resourceManager = helper.getResourceManager();
            File py = resourceManager.getResourceFiles().get("mil/sstaftest/simplemock/upper.py")
;
            helper.setArgs(List.of("python3", py.getAbsolutePath()));
            exerciseApplication(helper);
        });
    }
    @Test
    @DisplayName("Confirm that a durable AppSession can be created to run the 'upper.py' script")
    void toUpperTestDurable() {
        try {
            AppConfiguration config = AppConfiguration.builder()
                    .resourceOwner(SimpleMockFeature.class)
                    .resource("mil/sstaftest/simplemock/upper.py")
                    .mode(AppSupport.Mode.DURABLE)
                    .build();
            AppAdapter helper = AppSupport.createAdapter(config);
            ResourceManager resourceManager = helper.getResourceManager();
            File py = resourceManager.getResourceFiles().get("mil/sstaftest/simplemock/upper.py")
;
            helper.setArgs(List.of("python3", py.getAbsolutePath()));
            exerciseApplication(helper);
        } catch (SSTAFException ie) {
            ie.printStackTrace();
            Assertions.fail();
        }
    }
    @Test
    @DisplayName("Confirm that command line arguments work as expected for durable sessions")
    void argsDurable() {
        assertDoesNotThrow(() -> {
            AppConfiguration config = AppConfiguration.builder()
                    .resourceOwner(SimpleMockFeature.class)
                    .resource("mil/sstaftest/simplemock/args.py")
                    .mode(AppSupport.Mode.DURABLE)
                    .build();
            AppAdapter helper = AppSupport.createAdapter(config);
```

```
            checkHelper(helper);
        });
    }
    @Test
    @DisplayName("Confirm that command line arguments work as expected for transient sessions")
    void argsTransient() {
        assertDoesNotThrow(() -> {
            AppConfiguration config = AppConfiguration.builder()
                    .resourceOwner(SimpleMockFeature.class)
                    .resource("mil/sstaftest/simplemock/args.py")
                    .mode(AppSupport.Mode.TRANSIENT)
                    .build();
            AppAdapter helper = AppSupport.createAdapter(config);
            checkHelper(helper);
        });
    }
    private void checkHelper(AppAdapter helper) throws IOException {
        ResourceManager resourceManager = helper.getResourceManager();
        File py = resourceManager.getResourceFiles().get("mil/sstaftest/simplemock/args.py");
        final String bilbo = "Bilbo";
        helper.setArgs(List.of("python3", py.getAbsolutePath(), bilbo));
        AppSession session = helper.activate();
        String arg0 = session.invoke("0");
        Assertions.assertEquals(py.getAbsolutePath(), arg0);
        String arg1 = session.invoke("1");
        Assertions.assertEquals(bilbo, arg1);
        final String frodo = "Frodo";
        helper.setArgs(List.of("python3", py.getAbsolutePath(), frodo));
        session = helper.activate();
        arg0 = session.invoke("0");
        Assertions.assertEquals(py.getAbsolutePath(), arg0);
        arg1 = session.invoke("1");
        Assertions.assertEquals(frodo, arg1);
        final String sam = "Sam";
        session = helper.activate(List.of("python3", py.getAbsolutePath(), sam));
        arg0 = session.invoke("0");
        Assertions.assertEquals(py.getAbsolutePath(), arg0);
        arg1 = session.invoke("1");
        Assertions.assertEquals(sam, arg1);
    }
    private void exerciseApplication(AppAdapter helper) {
        int count = 0;
        long deltaT = 0;
        for (int i = 0; i < 10; ++i) {
            long start = System.nanoTime();
            try (AppSession appSession = helper.activate()) {
                assertDoesNotThrow(() -> {
                    List<String> strings = List.of("Banana",
                            "sdSJD",
                            "IamtheveryModelofaModernMajorGeneral",
                            "@#%#^%$&@#@GDFG$grt2@$%#F2453S",
                            "OneRingtoRuleThemAll,OneRingtoFindThem");
                    for (String s : strings) {
                        String bob = appSession.invoke(s);
                        //System.out.println(s + " --> " + bob);
                        Assertions.assertEquals(s.toUpperCase(Locale.ROOT), bob);
                    }
                });
            } catch (Exception e) {
                e.printStackTrace();
                Assertions.fail();
            }
            long stop = System.nanoTime();
            deltaT += stop - start;
            ++count;
        }
        double avg_nano = (double) deltaT / (double) count;
        double avg_milli = avg_nano / 1000000.;
```

```
        System.out.printf("Executed %d iterations in %d ns average = %f ns / %f ms\n", count,
                deltaT, avg_nano, avg_milli);
    }
}
```

A.192 SSTAF/src/framework/mil.sstaf.core/src/integrationTest/java/mil/sstaf/core/integration/FeatureManagerIntegrationTest.java

```
package mil.sstaf.core.integration;
import mil.sstaf.core.configuration.SSTAFConfiguration;
import mil.sstaf.core.entity.EntityHandle;
import mil.sstaf.core.entity.FeatureManager;
import mil.sstaf.core.features.FeatureConfiguration;
import mil.sstaf.core.features.FeatureSpecification;
import mil.sstaf.core.features.IntContent;
import mil.sstaf.core.features.StringContent;
import mil.sstaf.core.module.ModuleLayerDefinition;
import mil.sstaf.core.util.SSTAFException;
import org.junit.jupiter.api.*;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import java.io.File;
import java.nio.file.Path;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Set;
import static org.junit.jupiter.api.Assertions.assertNotNull;
import static org.junit.jupiter.api.Assertions.assertTrue;
public class FeatureManagerIntegrationTest {
    private static final Logger logger = LoggerFactory.getLogger(FeatureManagerIntegrationTest.cl
ass);
    @BeforeEach
    public void setup() {
        System.setProperty(SSTAFConfiguration.SSTAF_CONFIGURATION_PROPERTY,
                "src" + File.separator +
                        "integrationTest" + File.separator +
                        "resources" + File.separator +
                        "EmptyConfiguration.json");
    }
    @Nested
    @DisplayName("Basic Tests")
    public class InnerOne {
        @Test
        @DisplayName("Confirm that the FeatureManager can load a Feature graph from a Handler spe
cification")
        void canLoadAHandlerBySpec() {
            FeatureSpecification spec = FeatureSpecification.builder()
                    .featureName("Handler1")
                    .majorVersion(3).minorVersion(1).requireExact(false).build();
            EntityHandle entityHandle = EntityHandle.makeDummyHandle();
            List<FeatureSpecification> specificationList = List.of(spec);
            Map<String, FeatureConfiguration> configurations = new HashMap<>();
            FeatureManager featureManager = new FeatureManager(entityHandle, specificationList, c
onfigurations, 3);
            featureManager.init();
            assertTrue(featureManager.getSpecificationForHandler(StringContent.class).isPresent()
);
        }
        @Test
        @DisplayName("Confirm that the FeatureManager can load a Feature graph from an Agent spec
ification")
        void canLoadAnAgentBySpec() {
            FeatureSpecification spec = FeatureSpecification.builder()
                    .featureName("Agent1")
                    .majorVersion(3).minorVersion(1).requireExact(false).build();
            EntityHandle entityHandle = EntityHandle.makeDummyHandle();
            List<FeatureSpecification> specificationList = List.of(spec);
```

```
            Map<String, FeatureConfiguration> configurations = new HashMap<>();
            FeatureManager featureManager = new FeatureManager(entityHandle, specificationList, c
onfigurations, 3);
            featureManager.init();
            assertTrue(featureManager.getSpecificationForHandler(IntContent.class).isPresent());
        }
    }
    @Nested
    @DisplayName("Test ModuleLayer support")
    public class InnerTwo {
        @Test
        @DisplayName("Confirm that module and path lists work")
        public void test1() {
            FeatureSpecification spec = FeatureSpecification.builder()
                    .featureName("Pinky")
                    .majorVersion(13).minorVersion(0).requireExact(false).build();
            EntityHandle entityHandle = EntityHandle.makeDummyHandle();
            List<FeatureSpecification> specificationList = List.of(spec);
            Map<String, FeatureConfiguration> configurations = new HashMap<>();
            FeatureManager featureManager = null;
            Set<Path> paths = Set.of(Path.of("src/integrationTest/resources/modules/pinky"));
            Set<String> modules = Set.of("mil.sstaftest.mocks.pinky");
            ModuleLayerDefinition moduleLayerDefinition =
                    ModuleLayerDefinition.builder()
                            .modulePaths(paths)
                            .modules(modules)
                            .build();
            featureManager = new FeatureManager(entityHandle, moduleLayerDefinition, specificatio
nList, configurations, 3);
            featureManager.init();
            assertNotNull(featureManager);
            assertTrue(featureManager.getSpecificationForHandler(StringContent.class).isPresent()
);
        }
        @Test
        @DisplayName("Confirm that an exception is thrown when the module isn't found")
        public void test2() {
            Assertions.assertThrows(SSTAFException.class, () -> {
                FeatureSpecification spec = FeatureSpecification.builder()
                        .featureName("Pinky")
                        .majorVersion(13).minorVersion(0).requireExact(false).build();
                EntityHandle entityHandle = EntityHandle.makeDummyHandle();
                List<FeatureSpecification> specificationList = List.of(spec);
                Map<String, FeatureConfiguration> configurations = new HashMap<>();
                Set<Path> paths = Set.of(Path.of("src/integrationTest/resources/modules/pinky"));
                Set<String> modules = Set.of("mil.sstaftest.mocks.notreal");
                ModuleLayerDefinition moduleLayerDefinition =
                        ModuleLayerDefinition.builder()
                                .modulePaths(paths)
                                .modules(modules)
                                .build();
                FeatureManager featureManager = new FeatureManager(entityHandle, moduleLayerDefin
ition, specificationList, configurations, 3);
                featureManager.init();
                assertTrue(featureManager.getSpecificationForHandler(StringContent.class).isPrese
nt());
            });
        }
        @Test
        @DisplayName("Confirm that transitive resolution works")
        public void test3() {
            Map<String, FeatureConfiguration> configurations = new HashMap<>();
            FeatureConfiguration sc = FeatureConfiguration.builder().build();
            configurations.put("Echo", sc);
            configurations.put("Delta", sc);
            configurations.put("Charlie", sc);
            configurations.put("Bravo", sc);
            configurations.put("Alpha", sc);
```

```
            configurations.put("James Bond", sc);
            Assertions.assertDoesNotThrow(() -> {
                FeatureSpecification spec = FeatureSpecification.builder()
                        .featureName("James Bond")
                        .majorVersion(7).minorVersion(0).requireExact(false).build();
                EntityHandle entityHandle = EntityHandle.makeDummyHandle();
                List<FeatureSpecification> specificationList = List.of(spec);
                String userDir = System.getProperty("user.dir");
                Set<Path> paths = Set.of(
                        Path.of(userDir, "../../testFeatures/integration/mil.sstaftest.alpha/buil
d/libs"),
                        Path.of(userDir, "../../testFeatures/integration/mil.sstaftest.bravo/buil
d/libs"),
                        Path.of(userDir, "../../testFeatures/integration/mil.sstaftest.charlie/bu
ild/libs"),
                        Path.of(userDir, "../../testFeatures/integration/mil.sstaftest.delta/buil
d/libs"),
                        Path.of(userDir, "../../testFeatures/integration/mil.sstaftest.echo/build
/libs"),
                        Path.of(userDir, "../../testFeatures/integration/mil.sstaftest.jamesbond/
build/libs"));
                Set<String> modules = Set.of("mil.sstaftest.jamesbond");
                ModuleLayerDefinition moduleLayerDefinition =
                        ModuleLayerDefinition.builder()
                                .modulePaths(paths)
                                .modules(modules)
                                .build();
                FeatureManager featureManager = new FeatureManager(entityHandle, moduleLayerDefin
ition,
                        specificationList, configurations, 3);
                featureManager.init();
            });
        }
    }
}
```

A.193 SSTAF/src/framework/mil.sstaf.core/src/integrationTest/java/mil/sstaf/core/integration/JsonReferenceProcessorTest.java

```
package mil.sstaf.core.integration;
public class JsonReferenceProcessorTest {

}
```

A.194 SSTAF/src/framework/mil.sstaf.core/src/integrationTest/java/mil/sstaf/core/integration/ModuleLayerSupportTest.java

```
package mil.sstaf.core.integration;
import mil.sstaf.core.module.ModuleLayerDefinition;
import mil.sstaf.core.module.ModuleLayerSupport;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Nested;
import org.junit.jupiter.api.Test;
import java.io.File;
import java.nio.file.Path;
import java.util.HashSet;
import java.util.Set;
import static org.junit.jupiter.api.Assertions.*;
public class ModuleLayerSupportTest {
    private ModuleLayer pl;
    private Path dir1;
    private Path dir2;
    @BeforeEach
    void before() {
        pl = ModuleLayer.boot();
        String dirBase = "src" + File.separator
                + "test" + File.separator
                + "resources" + File.separator;
        dir1 = Path.of("src" ,"integrationTest", "resources", "modules", "jamesbond");
        dir2 = Path.of("src" ,"integrationTest", "resources", "modules", "pinky");
    }
    @Nested
    @DisplayName("Test the happy path")
    class HappyTests {
        @Test
        @DisplayName("Confirm that if no modules are provided the parent layer is returned")
        public void fallbackToParentLayer1() {
            ModuleLayerDefinition moduleLayerDefinition =
                    ModuleLayerDefinition.builder().build();
            ModuleLayer ml = ModuleLayerSupport.makeModuleLayer(pl,
                    moduleLayerDefinition,
                    this.getClass().getClassLoader());
            assertEquals(pl, ml);
            //fail("Bob");
        }
        @Test
        @DisplayName("Confirm a new ModuleLayer can be made.")
        public void npn() {
            String dirBase = "src" + File.separator
                    + "test" + File.separator
                    + "resources" + File.separator;
            Set<Path> paths = Set.of(dir1);
            Set<String> modules = Set.of("mil.sstaftest.jamesbond");
            ModuleLayerDefinition moduleLayerDefinition =
                    ModuleLayerDefinition.builder()
                            .modulePaths(paths)
                            .modules(modules)
                            .build();
            ModuleLayer ml = ModuleLayerSupport.makeModuleLayer(pl,
                    moduleLayerDefinition,
                    this.getClass().getClassLoader());
            assertNotEquals(pl, ml);
            //fail("Bob");
        }
    }
}
```

A.195 SSTAF/src/framework/mil.sstaf.core/src/integrationTest/java/mil/sstaf/core/integration/ResolverIntegrationTest.java

```
package mil.sstaf.core.integration;
import mil.sstaf.core.entity.EntityHandle;
import mil.sstaf.core.features.*;
import mil.sstaf.core.configuration.SSTAFConfiguration;
import mil.sstaf.core.util.SSTAFException;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
import java.io.File;
import java.util.HashMap;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.ConcurrentMap;
class ResolverIntegrationTest {
    @BeforeEach
    public void setup() {
        System.setProperty(SSTAFConfiguration.SSTAF_CONFIGURATION_PROPERTY,
                "src" + File.separator +
                        "integrationTest" + File.separator +
                        "resources" + File.separator +
                        "EmptyConfiguration.json");
    }
    @Test
    @DisplayName("Confirm that generateServiceReport() provides a useful report")
    public void testGenerateServiceReport() {
        Assertions.assertDoesNotThrow(() -> {
            String report = Loaders.generateServiceReport(Feature.class);
            Assertions.assertNotNull(report);
            System.out.println(report);
            Assertions.assertEquals(8, countMatches(report, "Implemented"));
            Assertions.assertEquals(1, countMatches(report, "mil.sstaftest.mocks.pinky.Pinky"));
            Assertions.assertEquals(2, countMatches(report, "[mil.sstaftest.mocks.pinky.Pinky]"))
;
        });
    }
    private int countMatches(final String source, final String word) {
        String[] temp = source.split("\\s+");
        int count = 0;
        for (String s : temp) {
            if (word.equals(s))
                count++;
        }
        return count;
    }
    @Test
    @DisplayName("Confirm that multiple services with cross-linked @Requires can be resolved and
loaded")
    void multilayerLoadWithCircularRequirementWorks() {
        FeatureSpecification jbSpec = FeatureSpecification.builder().featureName("James Bond")
                .majorVersion(7).minorVersion(0).requireExact(false).build();
        ConcurrentMap<FeatureSpecification, Feature> featureCache = new ConcurrentHashMap<>();
        Map<String, FeatureConfiguration> configurationMap = new HashMap<>();
        for (String s : new String[]{"Echo", "Delta", "Charlie","Bravo","Alpha", "James Bond"}) {
            configurationMap.put(s, FeatureConfiguration.builder().build());
        }
        EntityHandle eh = EntityHandle.makeDummyHandle();
        Resolver resolver = new Resolver(featureCache, configurationMap, eh, 31415, ModuleLayer.b
oot());
        Agent jamesBond = (Agent) resolver.loadAndResolveDependencies(jbSpec);
        Assertions.assertDoesNotThrow(jamesBond::init);
    }
```

```
    @Test
    @DisplayName("Force a Resolver failure and confirm the error message is correct")
    void testResolverFailure1() {
        FeatureSpecification tomBombadil = FeatureSpecification.builder().featureName("Tom Bombad
il")
                .majorVersion(1).minorVersion(0).requireExact(false).build();
        Resolver resolver = Resolver.makeTransientResolver();
        SSTAFException boom = Assertions.assertThrows(SSTAFException.class,
                () -> resolver.loadAndResolveDependencies(tomBombadil));
        String message = boom.getMessage();
        Assertions.assertTrue(message.contains("Tom Bombadil"));
        Assertions.assertTrue(message.contains("Feature"));
    }
}
```

A.196 SSTAF/src/framework/mil.sstaf.core/src/integrationTest/java/mil/sstaf/core/integration/ResourceManagerTest.java

```
package mil.sstaf.core.integration;
import mil.sstaf.core.features.ResourceManager;
import mil.sstaftest.simplemock.SimpleMockFeature;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Nested;
import org.junit.jupiter.api.Test;
import java.io.File;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.Map;
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertTrue;
public class ResourceManagerTest {
    @Nested
    @DisplayName("ResourceManager tests that use the assembled jar")
    class ITests {
        @Test
        @DisplayName(value = "Confirm that providing a null resource owner throws NullPointerExce
ption")
        public void nullResourceOwnerThrows() {
            Assertions.assertThrows(NullPointerException.class, () -> {
                Path tempDir = Files.createTempDirectory("Banana");
                ResourceManager.extractResource(null, "extractionTest/extractMe.txt", tempDir);
            });
        }
        @Test
        @DisplayName(value = "Confirm that providing a null resource name throws NullPointerExcep
tion")
        public void nullResourceNameThrows() {
            Assertions.assertThrows(NullPointerException.class, () -> {
                Path tempDir = Files.createTempDirectory("Banana");
                ResourceManager.extractResource(this.getClass(), null, tempDir);
            });
        }
        @Test
        @DisplayName(value = "Confirm that providing a null directory throws NullPointerException
")
        public void nullDirectoryThrows() {
            Assertions.assertThrows(NullPointerException.class,
                    () -> ResourceManager.extractResource(this.getClass(),
                            "extractionTest/extractMe.txt", null));
        }

        @Test
        @DisplayName("Confirm that the constructor works and the checksum table can be read")
        public void constructorTest() {
            ResourceManager rm = ResourceManager.getManager(SimpleMockFeature.class);
            Assertions.assertEquals(2, rm.getResourceFiles().size());
            Assertions.assertEquals(2, rm.getNumWritten());
            Assertions.assertTrue(rm.getDirectory().toString().contains("SimpleMockFeature"));
        }
        @Test
        @DisplayName("Confirm that reusing the same directory results in no files being written")
        public void reusedDirTest() {
            ResourceManager rm = ResourceManager.getManager(SimpleMockFeature.class);
            Path where = rm.getDirectory();
            assertEquals(2, rm.getNumWritten());
            ResourceManager rm2 = ResourceManager.getManager(SimpleMockFeature.class, where);
            assertEquals(where.toString(), rm2.getDirectory().toString());
            assertEquals(0, rm2.getNumWritten());
            Map<String, File> map = rm2.getResourceFiles();
```

```
            for (Map.Entry<String, File> entry : map.entrySet()) {
                assertTrue(entry.getValue().exists());
                assertTrue(entry.getValue().canRead());
                assertTrue(entry.getValue().getPath().contains(entry.getKey()));
            }
        }
    }
}
```

A.197 SSTAF/src/framework/mil.sstaf.core/src/integrationTest/java/mil/sstaf/core/integration/SoldierIntegrationTest.java

```
package mil.sstaf.core.integration;
import mil.sstaf.core.entity.Soldier;
import mil.sstaf.core.configuration.SSTAFConfiguration;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
import java.io.File;
public class SoldierIntegrationTest {
    @BeforeEach
    public void setup() {
        System.setProperty(SSTAFConfiguration.SSTAF_CONFIGURATION_PROPERTY,
                "src" + File.separator +
                        "integrationTest" + File.separator +
                        "resources" + File.separator +
                        "EmptyConfiguration.json");
    }
    @Test
    @DisplayName("Confirm that a Soldier can load")
    void test1() {
        Soldier soldier = Soldier.from(new File("src/integrationTest/resources/soldier/TestSoldie
r1.json"));
        Assertions.assertNotNull(soldier);
    }
}
```

A.198 SSTAF/src/framework/mil.sstaf.core/src/integrationTest/java/module-info.java

```
open module mil.sstaf.core.integration {
    exports mil.sstaf.core.integration;
    requires mil.sstaf.core;

    requires org.slf4j;
    requires mil.sstaftest.simplemock;
    requires org.junit.jupiter.api;
    requires mil.sstaftest.echo;
    requires mil.sstaftest.delta;
    requires mil.sstaftest.charlie;
}
```

A.199 SSTAF/src/framework/mil.sstaf.core/src/integrationTest/resources/EmptyConfiguration.json

```
{
  "class" : "mil.sstaf.core.configuration.SSTAFConfiguration",
  "moduleLayerDefinition" : {
    "modules": [
    ],
    "modulePaths": [
    ]
  }
}
```

A.200 SSTAF/src/framework/mil.sstaf.core/src/integrationTest/resources/log4j2-test.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="TRACE">
    <Appenders>
        <Console name="Console" target="SYSTEM_OUT">
            <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"/>
        </Console>
    </Appenders>
    <Loggers>
        <Root level="info">
            <AppenderRef ref="Console"/>
        </Root>
    </Loggers>
</Configuration>
```

A.201 SSTAF/src/framework/mil.sstaf.core/src/integrationTest/resources/soldier/TestSoldier1.json

```
{
  "class" : "mil.sstaf.core.entity.Soldier",
  "name": "Test Dude",
  "rank": "E6",
  "moduleLayerDefinition" : {
    "modules": [
      "mil.sstaftest.jamesbond"
    ],
    "modulePaths": [
      "src/integrationTests/resources/modules/jamesbond"
    ]
  },
  "features": [
    {
      "featureName": "James Bond",
      "majorVersion": 7,
      "minorVersion": 0
    }
  ],
  "configurations": {
    "James Bond": {
      "class" : "mil.sstaf.core.features.FeatureConfiguration"
    },
    "Alpha": {
      "class" : "mil.sstaf.core.features.FeatureConfiguration"
    },
    "Bravo": {
      "class" : "mil.sstaf.core.features.FeatureConfiguration"
    },
    "Charlie": {
      "class" : "mil.sstaf.core.features.FeatureConfiguration"
    },
    "Delta": {
      "class" : "mil.sstaf.core.features.FeatureConfiguration"
    },
    "Echo": {
      "class" : "mil.sstaf.core.features.FeatureConfiguration"
    }
  }
}
```

A.202 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/configuration/Configuration.java

```
package mil.sstaf.core.configuration;
/**
 * Marker interface for the {@code Feature} configuration objects.
 */
public interface Configuration {
}
```

A.203 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/configuration/SSTAFConfiguration.java

```
package mil.sstaf.core.configuration;

import com.fasterxml.jackson.annotation.JsonIgnore;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import com.fasterxml.jackson.databind.JsonNode;
import lombok.Builder;
import lombok.Getter;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import mil.sstaf.core.json.JsonLoader;
import mil.sstaf.core.module.ModuleLayerDefinition;
import mil.sstaf.core.module.ModuleLayerSupport;
import mil.sstaf.core.util.SSTAFException;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.io.File;
import java.nio.file.Path;
import java.util.Arrays;
import java.util.concurrent.atomic.AtomicReference;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;


/**
 * Contains global configuration information for the SSTAF environment.
 *
 * {@code SSTAFConfiguration} provides a global environment for SSTAF, through
 * which other SSTAF components can access system-wide settings.
 *
 * The motivation for {@code SSTAFConfiguration} was as to provide the ability
 * to creatie a root {@link ModuleLayer} for all SSTAF components. This root
 * layer is used to load SSTAF plugin modules and is the parent layer for the
 * {@code ModuleLayer}s that can be created for each {@code Entity}.
 *
 * The location of the {@code SSTAFConfiguration} is specified through either
 * the {@code mil.sstaf.configuration} JVM property, or the
 * {@code SSTAF_CONFIGURATION} shell environment variable. The JVM property
 * takes precedence.
 */
@SuperBuilder
@Jacksonized
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
public class SSTAFConfiguration {

    private static final Logger logger = LoggerFactory.getLogger(SSTAFConfiguration.class);

    public static final String SSTAF_CONFIGURATION_PROPERTY = "mil.sstaf.configuration";
    public static final String SSTAF_CONFIGURATION_ENV = "SSTAF_CONFIGURATION";
    public static boolean requireConfiguration = false;

    @Getter
    @Builder.Default
    @JsonIgnore
    ModuleLayer rootLayer = null;

    @Getter
    private ModuleLayerDefinition moduleLayerDefinition;

    private static final Lock lock = new ReentrantLock();
    private static final AtomicReference<SSTAFConfiguration> instance = new AtomicReference<>();
```

```
    /**
     * Provides the {@code SSTAFConfiguration}, loading it if necessary.
     *
     *  @return the SSTAFConfiguration
     */
    public static SSTAFConfiguration getInstance() {
        lock.lock();
        SSTAFConfiguration sstafConfiguration = instance.get();
        if (sstafConfiguration == null) {
            sstafConfiguration = loadConfiguration();
        }
        lock.unlock();
        return sstafConfiguration;
    }

    private static void setInstance(SSTAFConfiguration config) {
        lock.lock();
        instance.set(config);
        lock.unlock();
    }

    private static boolean isFileValid(String configFileName) {
        if (configFileName != null && configFileName.length() > 0) {
            File file = new File(configFileName);
            return file.exists() && file.canRead();
        }
        return false;
    }

    static SSTAFConfiguration loadConfiguration() {
        File f;
        String defaultFilename = System.getProperty("user.home") + File.separator +
                "SSTAFData" + File.separator + "config.json";
        String[] locations = {
                System.getProperty(SSTAF_CONFIGURATION_PROPERTY),
                System.getenv(SSTAF_CONFIGURATION_ENV),
                defaultFilename};
        SSTAFConfiguration sstafConfiguration = null;
        for (String filename : locations) {
            if (isFileValid(filename)) {
                sstafConfiguration = from(new File(filename));
                break;
            }
        }
        if (SSTAFConfiguration.requireConfiguration) {
            throw new SSTAFException(
                    String.format("Could not find a SSTAF configuration file in: %s",
                            Arrays.toString(locations)));
        } else if (sstafConfiguration == null) {
            // 2022-04-12 : RAB : Make a default empty configuration.
            sstafConfiguration = SSTAFConfiguration.builder().build();
            sstafConfiguration.init();
        }
        instance.set(sstafConfiguration);
        return sstafConfiguration;
    }

    /**
     * Clears the SSTAFConfiguration
     */
    static void reset() {
        lock.lock();
        instance.set(null);
        lock.unlock();
    }

    protected void init() {
        rootLayer = ModuleLayerSupport.makeModuleLayer(ModuleLayer.boot(),
```

```
                moduleLayerDefinition, this.getClass().getClassLoader());
    }

    private static SSTAFConfiguration from(File file) {
        JsonLoader loader = new JsonLoader();
        SSTAFConfiguration config = (SSTAFConfiguration) loader.load(Path.of(file.getAbsolutePath
()));
        config.init();
        return config;
    }

    static SSTAFConfiguration from(JsonNode node, Path sourceFile) {
        JsonLoader loader = new JsonLoader();
        SSTAFConfiguration config = (SSTAFConfiguration) loader.load(node, sourceFile);
        config.init();
        setInstance(config);
        return config;
    }

}
```

A.204 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/configuration/package-info.java

```
/**
 * This package provides support for system-level configuration of the
 * SSTAF environment.
 *
 * @since 1.0
 * @author Ron Bowers
 * @version 1.0
 */
package mil.sstaf.core.configuration;
```

A.205 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/entity/Address.java

```
package mil.sstaf.core.entity;
import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import lombok.Builder;
import lombok.EqualsAndHashCode;
import mil.sstaf.core.features.Agent;
import mil.sstaf.core.features.Handler;
import java.util.Comparator;
import java.util.Objects;
/**
 * @author Ron Bowers
 * @version 1.0
 * @see mil.sstaf.core.entity.FeatureManager
 * @see MessageDriven
 * @since 1.0
 * <p>
 * Specifies an address that can be used to route a {@link Message}.
 * </p>
 * <p>
 * A {@code Message} in SSTAF can be routed to a specific {@link Handler} or {@link Agent} within
a
 * specific {@link BaseEntity}. To accomplish this, the {@code Address} contains an {@link Entity
Handle} to specify the
 * {@code Entity} and a {@code String} to specify the name of the {@code Handler}. {@code Address
es} are used to
 * refer both to the source and destination of a {@code Message}.
 * </p>
 * <p>There are two flavors of {@code Address}, <em>internal</em> and <em>external</em>. An inter
nal {@code Address}
 * specifies only a {@code Handler} name. A {@code Message} that uses an internal {@code Address}
object as
 * its destination will be retained within the source {@code Entity} and routed directly to the
 * destination {@code Handler}. Sending {@code Message}s within an {@code Entity} is the preferre
d way of invoking
 * an asynchronous command or query within an {@code Entity}. An external {@code Address} contain
s an
 * {@code EntityHandle}. A {@code Message} that uses an external {@code Address}
 * object for its destination {@code Address} will be routed to the specified {@code Entity}. If
the {@code Address}
 * also contains a {@code Handler} name, the {@code Message} will be delivered to that {@code Han
dler}. Otherwise, the
 * {@code Message} will go to the {@code Handler} that accepts the content of the {@code Message}
.
 * </p>
 */
@Builder
@EqualsAndHashCode
@JsonIgnoreProperties("sourceDir")
public final class Address {
    /**
     * A convenient instance of {@code AddressComparator}
     */
    public static final Comparator<Address> COMPARATOR = new AddressComparator();
    //
    // Special addresses
    //
    /**
     * A special {@code Address} that, if used as a destination, causes a {@code Message} to be d
ropped.
     */
    public static final Address NOWHERE = Address.builder().entityHandle(null).handlerName("NOWHE
RE").build();
    /**
```

```
     * The handle to the {@code Entity} part of the address
     */
    public final EntityHandle entityHandle;
    /**
     * The name of the handler.
     */
    public final String handlerName;

    /**
     * <p>
     * Creates an {@code Address} that can be routed to a different {@code Entity}
     * than from which it originated.
     * </p><p>
     * Within the receiving Entity, the message will be dispatched to whichever {@code Handler}
     * responds to the contents of the {@code Message}.
     * </p>
     *
     * @param entityHandle the {@code EntityHandle} for the {@code Entity}
     * @return a new {@code Address}
     */
    public static Address makeExternalAddress(final EntityHandle entityHandle) {
        Objects.requireNonNull(entityHandle, "EntityHandle was null");
        return builder().entityHandle(entityHandle).handlerName(null).build();
    }
    /**
     * Creates an {@code Address} that is used to route messages within an {@code Entity}.
     *
     * @param handlerName the {@code Handler} that should receive the {@code Message}.
     * @return a new {@code Address}
     */
    public static Address makeInternalAddress(final String handlerName) {
        Objects.requireNonNull(handlerName, "HandlerName was null");
        return builder().entityHandle(null).handlerName(handlerName).build();
    }
    /**
     * Creates a fully-specified {@code Address} that can route messages either
     * internally or externally.
     *
     * @param entityHandle the {@code EntityHandle} for the Entity
     * @param handlerName  the {@code Handler} that should receive the message.
     * @return a new Address
     */
    public static Address makeAddress(final EntityHandle entityHandle, final String handlerName)
{
        Objects.requireNonNull(entityHandle, "EntityHandle was null");
        Objects.requireNonNull(handlerName, "HandlerName was null");
        return builder().entityHandle(entityHandle).handlerName(handlerName).build();
    }

    /**
     * Returns whether or not the {@code Address} is an internal-only address.
     *
     * @return true if the {@code Address} is internal
     */
    public boolean isInternal() {
        return entityHandle == null;
    }
    /**
     * Returns whether or not the {@code Address} is an external-capable address.
     * Note that an {@code Entity} can use an external address to talk to itself.
     *
     * @return true if the {@code Address} is an external address.
     */
    public boolean isExternal() {
        return entityHandle != null;
    }
    /**
     * Comparator for ordering Addresses
```

```
     */
    static class AddressComparator implements Comparator<Address> {
        /**
         * <p>Compares two {@code Address} objects.</p>
         *
         * <p>Internal {@code Address}es precede externals. If both have {@code EntityHandle}s th
ose are compared.
         * If the {@code EntityHandles}s are equal, the {@code Handler} names are compared.
         */
        @Override
        public int compare(Address o1, Address o2) {
            Objects.requireNonNull(o1, "Address must not be null");
            Objects.requireNonNull(o2, "Address must not be null");
            if (o1 == o2) {
                return 0;
            } else if (o1.isInternal() && o2.isExternal()) {
                return -1;
            } else if (o1.isExternal() && o2.isInternal()) {
                return 1;
            } else {
                int c = EntityHandle.comparator.compare(o1.entityHandle, o2.entityHandle);
                if (c == 0) {
                    String h1 = o1.handlerName;
                    String h2 = o2.handlerName;
                    if (h1 == null && h2 == null) return 0;
                    if (h2 == null) return -1;
                    if (h1 == null) return 1;
                    return Integer.compare(h1.compareTo(h2), 0);
                } else {
                    return c;
                }
            }
        }
    }
}
```

A.206 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/entity/BaseEntity.java

```
package mil.sstaf.core.entity;

import com.fasterxml.jackson.annotation.JsonIgnore;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.*;
import lombok.experimental.SuperBuilder;
import mil.sstaf.core.features.ExceptionCommand;
import mil.sstaf.core.features.FeatureConfiguration;
import mil.sstaf.core.features.FeatureSpecification;
import mil.sstaf.core.features.ProcessingResult;
import mil.sstaf.core.module.ModuleLayerDefinition;
import mil.sstaf.core.util.Injected;
import mil.sstaf.core.util.RNGUtilities;
import mil.sstaf.core.util.SSTAFException;
import org.apache.commons.math3.random.MersenneTwister;
import org.apache.commons.math3.random.RandomGenerator;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.util.*;
import java.util.concurrent.ConcurrentLinkedQueue;
import java.util.concurrent.atomic.AtomicLong;

/**
 * <p>The {@code Entity} class is the base class for all simulation participants.</p>
 * <p>{@code Entity} provides the core implementation for the {@code MessageDriven} architecture
by providing
 * the queues for receiving and returning messages and a mechanism for dispatching the messages t
o the appropriate
 * loaded {@code Handler}.</p>
 */
// 2022-04-07 : RAB : Fight the temptation add @Jacksonized. It can't be applied to abstract clas
ses.

@SuperBuilder
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
public abstract class BaseEntity implements Entity {

    /**
     * The {@code Logger} for this Entity.
     */
    private static final Logger logger = LoggerFactory.getLogger(BaseEntity.class);
    @Getter
    @Builder.Default
    protected UUID uuid = null;
    /**
     * A unique identification number associated with each {@code Entity}.
     */
    @Getter
    @Builder.Default
    @JsonIgnore
    // 2022-04-12 : RAB : Non-final so that subclasses can replace with custom IDs.
    protected long id = 0;
    /**
     * The random number generator for this {@code Entity}. This generator acts as the source
     * for the seeds used in any loaded {@code Feature}s.
     */
    @Builder.Default
    @JsonIgnore
    protected RandomGenerator randomGenerator = null;
    /**
```

```
     * The {@link FeatureManager} that manages the {@code Features} assigned to this {@code Entit
y}. The
     * {@code FeatureManager} holds the references to the {@code Feature} implementations, dispat
ches messages
     * to them, gathers responses and invokes the {@code tick()} method on implementations of {@c
ode Agent}
     */
    @Getter(AccessLevel.PACKAGE)
    @JsonIgnore
    protected FeatureManager featureManager;
    //
    // In and Out queues
    //
    @Builder.Default
    @JsonIgnore
    protected Queue<Message> inboundQueue = null;
    @Builder.Default
    @JsonIgnore
    protected Queue<Message> outboundQueue = null;
    @Injected(name = "randomSeed")
    protected long randomSeed;

    //
    // Identification
    //
    @Getter
    @Setter
    protected String name;

    @Getter
    @Builder.Default
    protected EntityHandle handle = null;

    @Getter
    @Setter
    @Builder.Default
    protected Force force = Force.GRAY;

    @Getter
    @Singular
    protected Map<String, FeatureConfiguration> configurations;

    @Getter
    @Singular
    protected List<FeatureSpecification> features;
    @Builder.Default
    @JsonIgnore
    private AtomicLong msgCounter = null;

    @Getter
    @Builder.Default
    @JsonIgnore
    private boolean initialized = false;

    @Getter
    private ModuleLayerDefinition moduleLayerDefinition;

    /**
     * Constructor
     *
     * @param builder the {@code Builder} to use to construct the {@code Entity}
     */
    protected BaseEntity(BaseEntity.BaseEntityBuilder<?, ?> builder) {
        logger.trace("Constructing Entity from builder {}", builder);
        this.id = BlockCounter.userCounter.getID();
        this.configurations = buildConfigurations(builder);
        uuid = UUID.randomUUID();
```

```
        this.name = builder.name == null || builder.name.length() == 0 ? this.uuid.toString() : b
uilder.name;

        this.randomSeed = builder.randomSeed == 0 ? id : builder.randomSeed;
        this.randomGenerator = new MersenneTwister();
        this.randomGenerator.setSeed(randomSeed);

        features = builder.features == null ? List.of() : builder.features;

        handle = new EntityHandle(this);
        featureManager = new FeatureManager(handle,
                moduleLayerDefinition,
                features,
                this.configurations,
                RNGUtilities.generateSubSeed(randomGenerator));
        if (logger.isDebugEnabled()) {
            logger.debug("Entity '{}' constructed, features = {}", name,
                    featureManager.generateConfigurationReport());
        }
        inboundQueue = new PriorityQueue<>(new MessageQueueComparator());
        outboundQueue = new ConcurrentLinkedQueue<>();
        msgCounter = new AtomicLong(0);
    }


    protected BaseEntity() {
        this.featureManager = null;
        this.uuid = UUID.randomUUID();
        this.name = uuid.toString();
        this.randomGenerator = null;
        this.id = -666L;
        inboundQueue = new PriorityQueue<>(new MessageQueueComparator());
        outboundQueue = new ConcurrentLinkedQueue<>();
        msgCounter = new AtomicLong(0);
    }

    protected BaseEntity(String name, long id) {
        this.featureManager = null;
        this.name = name;
        this.randomGenerator = null;
        this.id = id;
        this.handle = new EntityHandle(this);
        this.uuid = UUID.randomUUID();
        inboundQueue = new PriorityQueue<>(new MessageQueueComparator());
        outboundQueue = new ConcurrentLinkedQueue<>();
        msgCounter = new AtomicLong(0);
    }

    private Map<String, FeatureConfiguration> buildConfigurations(BaseEntityBuilder<?, ?> builder
) {
        Map<String, FeatureConfiguration> configMap;
        if (builder.configurations$key != null) {
            configMap = new TreeMap<>();
            for (int i = 0; i < builder.configurations$key.size(); ++i) {
                configMap.put(builder.configurations$key.get(i), builder.configurations$value.get
(i));
            }
        } else {
            configMap = Map.of();
        }
        return configMap;
    }

    /**
     * Injects an item into the features
     *
     * @param object the item to inject.
     */
```

```
    @Override
    public void injectInFeatures(Object object) {
        featureManager.injectAll(object);
    }


    /**
     * Checks to determine if the {@code Entity} has been initialized.
     * <p>
     * If the {@code Entity has not been }
     */
    @Override
    public void checkInit() {
        if (!initialized) {
            throw new SSTAFException("Entity " + name + " has not been initialized");
        }
    }

    /**
     * {@inheritDoc}
     */
    @Override
    public long generateSequenceNumber() {
        return msgCounter.getAndIncrement();
    }

    /**
     * {@inheritDoc}
     */
    @Override
    public List<Message> takeInbound() {
        checkInit();
        return takeFromQueue(inboundQueue);
    }

    /**
     * {@inheritDoc}
     */
    @Override
    public List<Message> takeOutbound() {
        checkInit();
        return takeFromQueue(outboundQueue);
    }

    private List<Message> takeFromQueue(Queue<Message> inboundQueue) {
        List<Message> out = new ArrayList<>(inboundQueue.size());
        out.addAll(inboundQueue);
        inboundQueue.clear();
        return out;
    }

    /**
     * Provides the current depth of the inbound message queue.
     *
     * @return the queue depth
     */
    @Override
    public int getInboundQueueDepth() {
        return inboundQueue.size();
    }

    /**
     * Runs all per-tick agents.
     * <p>
     * Agents can produce both outbound messages and internal
     *
     * @param currentTime_ms the current simulation time
     * @return the time of the next event
```

```
     */
    @Override
    public long runAgents(final long currentTime_ms) {
        checkInit();
        ProcessingResult pr = featureManager.runAllAgents(currentTime_ms);
        routeProcessingResults(pr);
        return getNextEventTime();
    }

    /**
     * Initializes the {@code Entity}.
     */
    @Override
    public void init() {
        logger.trace("Initializing Entity {}", name);
        if (handle == null) {
            throw new IllegalStateException("EntityHandle has not been initialized");
        }
        randomGenerator.setSeed(randomSeed);
        featureManager.init();
        initialized = true;
    }

    /**
     * Returns the time of the next event in the queue.
     *
     * @return Time of the next event or Double.POSITIVE_INFINITY if there is no event.
     */
    private long getNextEventTime() {
        Message message = inboundQueue.peek();
        if (message instanceof EntityEvent) {
            return ((EntityEvent) message).getEventTime_ms();
        } else {
            return Long.MAX_VALUE;
        }
    }

    /**
     * Polls the queue for the next event whose eventTime is less than or equal to the current si
mulation time.
     *
     * @param currentTime_ms the current simulation time
     * @return An Optional that contains the Event or is empty if no events are found.
     */
    private Optional<Message> getNextMessage(final long currentTime_ms) {
        Message message = inboundQueue.peek();
        if (message == null) {
            return Optional.empty();
        } else {
            if (message instanceof EntityEvent) {
                if (((EntityEvent) message).getEventTime_ms() > currentTime_ms) {
                    return Optional.empty();
                }
            }
        }
        return Optional.ofNullable(inboundQueue.poll());
    }

    /**
     * {@inheritDoc}
     */
    @Override
    public void receive(Message message) {
        checkInit();
        inboundQueue.offer(message);
        if (logger.isTraceEnabled()) {
            logger.trace("Entity {} received message: {} queue length: {}",
                    getName(), message, inboundQueue.size());
```

```
        }
    }

    /**
     * Answers whether the Entity has a {@code Handler} that can process the content.
     *
     * @param contentClass the {@code Class} to check
     * @return true if it can be handled, false otherwise.
     */
    @Override
    public boolean canHandle(Class<?> contentClass) {
        boolean ok = featureManager.canHandle(contentClass);
        if (!ok) {
            logger.warn("Entity {} can't process messages of type {}, no handler for that type wa
s specified in the input",
                    getName(), contentClass.getName());
        }
        return ok;
    }


    /**
     * Processes all messages in the queue up to the specified time.
     *
     * @param currentTime_ms the current simulation time.
     * @return the time of the next event in the queue.
     */
    @Override
    public long processMessages(final long currentTime_ms) {
        checkInit();
        logger.trace("Entity {}, starting tick at {}", getPath(), currentTime_ms);
        Optional<Message> optMessage;
        while ((optMessage = getNextMessage(currentTime_ms)).isPresent()) {
            Message message = optMessage.get();
            try {
                logger.trace("Entity {}, processing {} at {}", getPath(), message, currentTime_ms
);
                ProcessingResult pr = featureManager.process(message, currentTime_ms);
                logger.trace("Entity {}, result was {}", getPath(), pr);
                routeProcessingResults(pr);
            } catch (Exception e) {
                logger.error("Entity {}: {}", name, e);
                e.printStackTrace();
                String errMsg = "Entity " + name + ": Error at time " + currentTime_ms + " ms, pr
ocessing " + message;
                sendErrorResponse(message.getSequenceNumber(), errMsg, e, message.getRespondTo())
;
            }
            if (logger.isTraceEnabled()) {
                logger.trace("Entity {}, done processing message", getName());
            }
        }
        return getNextEventTime();
    }


    /**
     * Routes any messages within processing results to either the inbound or outbound queues acc
ording to their
     * destination address.
     *
     * @param pr the ProcessingResult
     */
    private void routeProcessingResults(ProcessingResult pr) {
        pr.messages.forEach(m -> {
            if (m.getDestination().equals(Address.NOWHERE)) {
                logger.trace("In {}, dropping message to NOWHERE from {}, contents = {}",
                        getName(), m.getSource().handlerName, m.getContent());
```

```
            } else if (m.getDestination().entityHandle.equals(this.handle)) {
                logger.trace("In {}, submitting local message from {} to {}, contents = {}",
                        getName(), m.getSource().handlerName, m.getDestination().handlerName, m.g
etContent());
                inboundQueue.offer(m);
            } else {
                logger.trace("In {}, submitting message from {} to {}:{}, contents = {}",
                        getName(), m.getSource().handlerName,
                        m.getDestination().entityHandle.getPath(),
                        m.getDestination().handlerName,
                        m.getContent());
                outboundQueue.offer(m);
            }
        });
    }

    /**
     * Builds an ErrorResponse
     *
     * @param id          the sequence number of the {@code Message} that resulted int the error
     * @param message     the error message
     * @param exception   the caught exception
     * @param destination where the message is going
     */
    @Override
    public void sendErrorResponse(final long id, final String message, final Throwable exception,
                                  final Address destination) {
        var b = ErrorResponse.builder()
                .source(Address.makeAddress(this.getHandle(), "NONE"))
                .destination(destination)
                .messageID(id)
                .sequenceNumber(this.generateSequenceNumber())
                .content(ExceptionCommand.builder().thrown(exception).build())
                .errorDescription(message);
        Message out = b.build();
        logger.trace("Entity {} sending {}", name, out);
        outboundQueue.offer(out);
    }

    static class Dummy extends BaseEntity {
        @Override
        public String getPath() {
            return "Dummy";
        }

    }
}
```

A.207 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/entity/Entity.java

```
package mil.sstaf.core.entity;
public interface Entity extends MessageDriven {
    /**
     * The delimiter to use between levels in the path.
     */
    String ENTITY_PATH_DELIMITER = ":";
    void injectInFeatures(Object object);
    EntityHandle getHandle();
    /**
     * Provides the current path (organizational hierarchy) for this {@code Entity}
     *
     * @return the path
     */
    String getPath();
    String getName();
    Force getForce();
    void setForce(Force force);
    void checkInit();
    int getInboundQueueDepth();
    long runAgents(long currentTime_ms);
    void init();
    boolean canHandle(Class<?> contentClass);
    long processMessages(long currentTime_ms);
    void sendErrorResponse(long id, String message, Throwable exception,
                           Address destination);

}
```

A.208 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/entity/EntityAction.java

```
package mil.sstaf.core.entity;
import lombok.Data;
import lombok.EqualsAndHashCode;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
/**
 * A {@code Message} that invokes an immediate action on the {@code Entity}.
 */
@EqualsAndHashCode(callSuper = true)
@Data
@SuperBuilder
@Jacksonized
public class EntityAction extends SimpleMessage {

}
```

A.209 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/entity/EntityEvent.java

```
package mil.sstaf.core.entity;
import lombok.Data;
import lombok.EqualsAndHashCode;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
/**
 * A {@code Message} that contains an event that the {@code Entity} will process at some simulati
on time.
 */
@EqualsAndHashCode(callSuper = true)
@Data
@SuperBuilder
@Jacksonized
public class EntityEvent extends EntityAction {
    private final long eventTime_ms;
}
```

A.210 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/entity/EntityHandle.java

```
package mil.sstaf.core.entity;
import com.fasterxml.jackson.databind.annotation.JsonSerialize;
import mil.sstaf.core.json.EntityHandleSerializer;
import java.util.Comparator;
import java.util.Objects;
/**
 * An indirect reference to an {@code Entity} that is used for message routing.
 * <p>
 * The {@code EntityHandle} provides the ability for the {@code Session}, as well as
 * {@code Handler}s and {@code Agent}s, to create {@code Message}s for other {@code Entities}
 * without granting them direct access to the {@code Entity}. Direct access is avoided
 * to prevent uncontrolled data access across threads.
 */
@JsonSerialize(using= EntityHandleSerializer.class)
final public class EntityHandle implements Comparable<EntityHandle> {
    public static final Comparator<EntityHandle> comparator = new MyComparator();
    private final Entity wrapped;
    /**
     * Constructor
     *
     * @param entity the {@code Entity} to which the handle refers
     */
    public EntityHandle(final Entity entity) {
        this.wrapped = Objects.requireNonNull(entity, "Entity must not be null");
    }
    private EntityHandle() {
        wrapped = new BaseEntity.Dummy();
    }
    public static EntityHandle makeDummyHandle() {
        return new EntityHandle();
    }
    /**
     * Provides the wrapped {@code Entity}
     *
     * @return the Entity
     */
    Entity getWrapped() {
        return wrapped;
    }
    /**
     * {@inheritDoc}
     */
    @Override
    public String toString() {
        return "id=" + wrapped.getId()
                + " forces=" + wrapped.getForce()
                + " path=" + wrapped.getPath()
                + " name=" + wrapped.getName();
    }
    public String getForcePath() {
        return getForce().name() + Entity.ENTITY_PATH_DELIMITER + getPath();
    }
    /**
     * Provides the Forces to which the Entity belongs.
     *
     * @return the Forces enum
     */
    public Force getForce() {
        return wrapped.getForce();
    }
    /**
     * Sets the force membership for the Entity.
```

```
     *
     * @param force the {@code Forces}
     */
    public void setForce(Force force) {
        wrapped.setForce(force);
    }
    /**
     * Provides the unique ID number for the Entity.
     *
     * @return the id number
     */
    public long getId() {
        return wrapped.getId();
    }
    /**
     * Provides the next message sequence number for messages originating from the
     * wrapped {@code Entity}
     *
     * @return the sequence number
     */
    public long getMessageSequenceNumber() {
        return wrapped.generateSequenceNumber();
    }
    /**
     * Provides the current path for this Entity.
     * <p>
     * For Soldiers, the path is the unit hierarchy and position within their containing Unit. Fo
r
     * other Entities it is simply their name. Since Soldiers can be moved between Units their pa
th
     * can change
     *
     * @return the path
     */
    public String getPath() {
        return wrapped.getPath();
    }
    /**
     * Provides the name of the Entity
     *
     * @return the name
     */
    public String getName() {
        return wrapped.getName();
    }
    /**
     * Returns a composite label that constists of the Entity path and name
     *
     * @return the composite label
     */
    public String getPathAndName() {
        String path = getPath();
        String name = getName();
        return (path == null ? "null" : path) + "[" + (name == null ? "null" : name) + "]";
    }
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        EntityHandle that = (EntityHandle) o;
        return Objects.equals(wrapped, that.wrapped);
    }
    /**
     * {@inheritDoc}
     */
    @Override
    public int hashCode() {
        return wrapped.hashCode();
```

```
    }
    /**
     * {@inheritDoc}
     */
    @Override
    public int compareTo(EntityHandle o) {
        return Long.compare(wrapped.getId(), o.wrapped.getId());
    }
    static class MyComparator implements Comparator<EntityHandle> {
        @Override
        public int compare(EntityHandle o1, EntityHandle o2) {
            if (o1 == o2) {
                return 0;
            } else if (o2 == null) {
                return -1;
            } else if (o1 == null) {
                return 1;
            } else {
                return o1.compareTo(o2);
            }
        }
    }
    /**
     * Answers whether or not the wrapped Entity has a {@code Handler} that can process the conte
nt.
     *
     * @param contentClass the content to check
     * @return true if it can be handled, false otherwise.
     */
    public boolean canHandle(Class<?> contentClass) {
        return wrapped.canHandle(contentClass);
    }
}
```

A.211 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/entity/EntityRegistry.java

```
package mil.sstaf.core.entity;
import java.util.*;
/**
 * Registry for all Entity instances in the SSTAF environment.
 * <p>
 * Entities are divided into two categories, simulation and system. Simulation entities
 * are those Entities that represent participants in the simulation. Examples include Soldiers an
d Units.
 * System entities are support elements within the environment.
 */
public class EntityRegistry {
    private final Map<Force, List<EntityHandle>> forcesMap = new EnumMap<>(Force.class);
    private final Map<Long, EntityHandle> allEntityHandles = new HashMap<>();
    // 2022.04.15 : RAB : Added direct Path to Handle mapping
    private final Map<String, EntityHandle> pathToHandleMap = new HashMap<>();
    private final Map<Long, Entity> allEntities = new HashMap<>();
    private final List<Entity> simulationEntities = new ArrayList<>();
    private Address clientAddress;

    public void registerEntities(Map<Force, List<BaseEntity>> participants) {
        for (var forceEntry : participants.entrySet()) {
            for (var entity : forceEntry.getValue()) {
                registerEntity(forceEntry.getKey(), entity);
            }
        }
    }
    /**
     * Registers an {@code Entity} and assigns it to the specified {@code Force}
     *
     * @param force  the {@code Force}
     * @param entity the {@code Entity}
     */
    public void registerEntity(final Force force, final Entity entity) {
        entity.setForce(force);
        EntityHandle entityHandle = entity.getHandle();
        List<EntityHandle> forceList = forcesMap.computeIfAbsent(force, forces1 -> new ArrayList<
>());
        forceList.add(entityHandle);
        if (entity instanceof Unit) {
            Unit unit = (Unit) entity;
            for (var s : unit.getDirectMembers().values()) {
                registerEntity(force, s);
            }
            for (var u : unit.getSubUnitMap().values()) {
                registerEntity(force, u);
            }
        }
    }
    /**
     * Traverse the Forces map and builds other Collections to
     * facilitate lookups.
     */
    public void compileEntityMaps() {
        for (Force force : Force.values()) {
            List<EntityHandle> entityHandleList = forcesMap.computeIfAbsent(force, force1 -> new
ArrayList<>());
            for (EntityHandle entityHandle : entityHandleList) {
                Entity entity = entityHandle.getWrapped();
                long id = entity.getId();
                allEntities.put(id, entity);
                allEntityHandles.put(id, entityHandle);
                String path = force.name() + Entity.ENTITY_PATH_DELIMITER + entity.getPath();
```

```
                pathToHandleMap.put(path, entityHandle);
                if (force != Force.SYSTEM) {
                    simulationEntities.add(entity);
                }
            }
        }
    }

    /**
     * Provides the {@code Address} for the client proxy
     *
     * @return the {@code Address}
     */
    public Address getClientAddress() {
        return clientAddress;
    }
    /**
     * Creates and sets the {@code Address} for the client proxy
     *
     * @param clientHandle the {@code EntityHandle} for the client.
     */
    public void setClientAddress(EntityHandle clientHandle) {
        this.clientAddress = Address.makeExternalAddress(clientHandle);
    }
    /**
     * Provides all the entities
     *
     * @return an unmodifiable collection of all the registered entities.
     */
    public Collection<Entity> getAllEntities() {
        return Collections.unmodifiableCollection(allEntities.values());
    }
    /**
     * Provides all the simulation entities
     *
     * @return an unmodifiable
     */
    public Collection<Entity> getSimulationEntities() {
        return Collections.unmodifiableCollection(simulationEntities);
    }
    /**
     * Provides all of the {@code EntityHandle}s
     *
     * @return an unmodifiable collection of EntityHandles
     */
    public Collection<EntityHandle> getAllEntityHandles() {
        return Collections.unmodifiableCollection(allEntityHandles.values());
    }
    /**
     * Provides the Entity associated with the given handle
     *
     * @param handle the {@code EntityHandle} to dereference
     * @return an Optional that contains the Entity or is empty if the EntityHandle did not wrap
an Entity.
     */
    public Optional<Entity> getEntityByHandle(EntityHandle handle) {
        Objects.requireNonNull(handle, "EntityHandle must not be null");
        return Optional.ofNullable(handle.getWrapped());
    }
    /**
     * Provides the EntityHandle associated with the given id
     *
     * @param id the id
     * @return an Optional that contains the EntityHandle or is empty if the id did not correspon
d to an Entity
     */
    public Optional<EntityHandle> getHandle(final Long id) {
        if (allEntityHandles.containsKey(id)) {
```

```
            return Optional.ofNullable(allEntityHandles.get(id));
        } else {
            return Optional.empty();
        }
    }
    /**
     * Provides the EntityHandle associated with the given path string
     *
     * @param path the path to the entity
     * @return an Optional that contains the EntityHandle or is empty if the path did not corresp
ond to an Entity
     */
    public Optional<EntityHandle> getHandle(final String path) {
        String fullPath;
        boolean hasForcePrefix = false;
        if (path.contains(":")) {
            for (Force f : Force.values()) {
                if (path.startsWith(f.name())) {
                    hasForcePrefix = true;
                    break;
                }
            }
            if (!hasForcePrefix) {
                fullPath = Force.BLUE.name() + Entity.ENTITY_PATH_DELIMITER + path;
            } else {
                fullPath = path;
            }
        } else {
            fullPath = Force.BLUE.name() + Entity.ENTITY_PATH_DELIMITER + path;
        }
        if (pathToHandleMap.containsKey(fullPath)) {
            return Optional.ofNullable(pathToHandleMap.get(fullPath));
        } else {
            return Optional.empty();
        }
    }
    /**
     * Provides the Entity associated with the given id
     *
     * @param id the id
     * @return an Optional that contains the Entity or is empty if the id did not correspond to a
n Entity.
     */
    public Optional<Entity> getEntity(final Long id) {
        if (allEntities.containsKey(id)) {
            return Optional.ofNullable(allEntities.get(id));
        } else {
            return Optional.empty();
        }
    }
}
```

A.212 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/entity/ErrorResponse.java

```
package mil.sstaf.core.entity;
import lombok.Data;
import lombok.EqualsAndHashCode;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
/**
 * Message for conveying than an error has occurred.
 */
@EqualsAndHashCode(callSuper = true)
@Data
@SuperBuilder
@Jacksonized
public final class ErrorResponse extends MessageResponse {
    private final String errorDescription;
    public Throwable getThrowable() {
        Object content = getContent();
        if (content instanceof Throwable) return (Throwable) content;
        else return null;
    }
}
```

A.213 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/entity/FeatureManager.java

```
package mil.sstaf.core.entity;

import mil.sstaf.core.configuration.SSTAFConfiguration;
import mil.sstaf.core.features.*;
import mil.sstaf.core.module.ModuleLayerDefinition;
import mil.sstaf.core.module.ModuleLayerSupport;
import mil.sstaf.core.util.*;
import org.apache.commons.math3.random.MersenneTwister;
import org.apache.commons.math3.random.RandomGenerator;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.util.*;
import java.util.concurrent.atomic.AtomicLong;

/**
 * Manages all Features (Providers, Handlers and Agents) associated with an Entity
 */
public class FeatureManager {

    private static final Logger logger = LoggerFactory.getLogger(FeatureManager.class);
    private final Map<FeatureSpecification, Feature> features = new HashMap<>();
    private final Map<Class<?>, Handler> contentToHandlerMap = new HashMap<>();
    private final Map<String, Handler> nameToHandlerMap = new HashMap<>();
    private final Set<Agent> agents = new HashSet<>();
    //
    // Needed for return address in Agent messages.
    //
    private final EntityHandle owner;
    private final AtomicLong messageCounter = new AtomicLong(0);
    //
    // ModuleLayer support
    //
    ModuleLayer moduleLayer;
    private boolean initialized = false;

    /**
     * Simplified constructor.
     *
     * <p>This constructor does not provide the ability to add modules or define module paths.</p
>
     *
     * @param owner          the {@code Entity} to which these features belong.
     * @param features       A list of the features (Handlers or Agents) that the {@code Entity}
provides
     * @param configurations the set of configuration objects provided to the Entity, as a JSONOb
ject
     * @param randomSeed     a seed for random number generation.
     */
    public FeatureManager(EntityHandle owner, List<FeatureSpecification> features,
                          Map<String, FeatureConfiguration> configurations, long randomSeed) {
        this(owner, null, features, configurations, randomSeed);
    }

    /**
     * Constructor
     *
     * @param owner          the {@code Entity} to which these features belong.
     * @param moduleLayerDefinition the configuration for a new {@code ModuleLayer}
     * @param features       A list of the features (Handlers or Agents) that the {@code Entity}
provides
```

```
     * @param configurations the set of configuration objects provided to the Entity, as a JSONOb
ject
     * @param randomSeed     a seed for random number generation.
     */
    public FeatureManager(EntityHandle owner,
                          ModuleLayerDefinition moduleLayerDefinition,
                          List<FeatureSpecification> features,
                          Map<String, FeatureConfiguration> configurations,
                          long randomSeed) {

        //
        // The simple stuff :-)
        //
        this.owner = owner;
        RandomGenerator generator = new MersenneTwister(randomSeed);

        //
        // If modules and paths have been configured, set up a new module layer within this entit
y.
        // This new layer will encapsulate all the features loaded into this entity.
        // Otherwise, use the class loader associated with the Entity itself.
        //
        moduleLayer = ModuleLayerSupport.makeModuleLayer(SSTAFConfiguration.getInstance().getRoot
Layer(),
                moduleLayerDefinition, owner.getClass().getClassLoader());

        if (!features.isEmpty()) {
            logger.debug("Resolving features");
            Resolver resolver = new Resolver(this.features, configurations, owner,
                    RNGUtilities.generateSubSeed(generator), moduleLayer);

            features.forEach(spec -> {
                logger.debug("Resolving {}", spec);
                Feature f = resolver.loadAndResolveDependencies(spec);
                logger.debug("Got {} / {}", f.getName(), f.getDescription());
                if (f instanceof Agent) {
                    register((Agent) f);
                } else if (f instanceof Handler) {
                    register((Handler) f);
                } else {
                    register(f);
                }
            });
        }
    }

    /**
     * Empty constructor for testing
     */
    public FeatureManager() {
        this(null, List.of(), Map.of(), 1234567);
    }

    /**
     * Confirms that the object has been initialized properly. If not, a
     * {@code SSTAFException) is thrown
     */
    private void checkInit() {
        if (!initialized) {
            throw new SSTAFException("Handlers has not been properly initialized");
        }
    }

    /**
     * Processes a single message from a client.
     *
     * @param message        the message to be processed.
     * @param currentTime_ms the current simulation time
```

```
     * @return a {@code ProcessingResult} that encapsulates the result of processing the message.
     */
    public ProcessingResult process(final Message message,
                                    final long currentTime_ms) {
        checkInit();

        final HandlerContent content = message.getContent();

        if (content == null) {
            var b = ErrorResponse.builder();
            b.source(Address.makeAddress(owner, "FeatureManager"));
            b.errorDescription("Message content was null");
            b.destination(message.getRespondTo());
            b.sequenceNumber(messageCounter.getAndIncrement());
            b.messageID(message.getSequenceNumber());
            logger.error("In {}, message content was null", owner.getPath());
            return ProcessingResult.of(b.build());
        }

        final long scheduledTime_ms = (message instanceof EntityEvent) ?
                ((EntityEvent) message).getEventTime_ms() : currentTime_ms;

        //
        // Select handler based on destination name and content class
        //
        Handler handler = getHandler(content.getClass(), message.getDestination().handlerName);

        if (handler == null) {
            if (content instanceof ExceptionCommand) {
                //
                // default handling for error messages.
                //
                Throwable t = ((ExceptionCommand) content).getThrown();
                logger.error("Received throwable from " + message.getSource().toString(), t);
                return ProcessingResult.empty();
            } else {
                var b = ErrorResponse.builder();
                b.source(Address.makeAddress(owner, "FeatureManager"));
                b.destination(message.getRespondTo());
                b.sequenceNumber(messageCounter.getAndIncrement());
                b.messageID(message.getSequenceNumber());
                String msg = "No handler for message of type " + content.getClass().getName();
                b.errorDescription(msg);
                b.content(ExceptionCommand.builder().thrown(new SSTAFException(msg)).build());
                logger.error(
                        "\tIn {}, No handler for message of type {}, received from {}\n" +
                                "\tTypes handled are {}\n" +
                                "\tThe handlers are {}",
                        owner.getPath(),
                        content.getClass(),
                        message.getSource(),
                        contentToHandlerMap.keySet(),
                        nameToHandlerMap.values());
                return ProcessingResult.of(b.build());
            }
        } else {
            logger.trace("Dispatching {} to {}", message, handler.getName());
            return handler.process(content, scheduledTime_ms, currentTime_ms,
                    message.getSource(), message.getSequenceNumber(), message.getRespondTo());
        }
    }

    /**
     * Chooses a Handler to process the message
     *
     * @param contentClass the Class of the message content.
     * @param handlerName  the name of the destination Handler specified in the message.
     * @return a Handler, or null if no suitable Handler was found.
```

```
     */
    Handler getHandler(final Class<? extends HandlerContent> contentClass, final String handlerNa
me) {
        Objects.requireNonNull(contentClass, "Content class was null");
        Handler handler;
        if (handlerName == null) {
            logger.trace("In {}, selecting handler using class {}",
                    owner.getPath(), contentClass.getName());
            handler = contentToHandlerMap.get(contentClass);

            if (handler == null) {
                for (Class<?> key : contentToHandlerMap.keySet()) {
                    if (key.getName().equals(contentClass.getName())) {
                        logger.error("Duplicated class! {} key CL = {}, message CL = {} ",
                                contentClass.getName(),
                                key.getClassLoader(),
                                contentClass.getClassLoader());
                    }
                }
            }
        } else {
            logger.trace("In {}, selecting handler using destination name {}",
                    owner.getPath(), handlerName);
            handler = nameToHandlerMap.get(handlerName);
            if (handler != null && !handler.contentHandled().contains(contentClass)) {
                handler = null;
                logger.debug("In {}, handler {} does not support {}",
                        owner.getPath(), handlerName, contentClass.getName());
            }
        }
        logger.debug("In {}, selected {} to handle {}", owner.getPath(),
                handler == null ? "null" : handler.getName(), contentClass.getName());
        logger.debug("In {}, module for message {}, classloader for module {}",
                owner.getPath(),
                contentClass.getModule().getName(),
                contentClass.getModule().getClassLoader());
        return handler;
    }

    /**
     * Generates a formatted String that describes the loaded features
     *
     * @return a String
     */
    String generateConfigurationReport() {
        StringBuilder sb = new StringBuilder();
        for (Feature f : features.values()) {
            sb.append('\'').append(f.getName()).append("' ")
                    .append(f.getMajorVersion()).append('.')
                    .append(f.getMinorVersion()).append('.')
                    .append(f.getPatchVersion()).append(' ');
            if (f instanceof Handler) {
                Handler h = (Handler) f;
                List<Class<? extends HandlerContent>> lc = h.contentHandled();
                if (lc == null) {
                    logger.error("Handler {} returned 'null' from contentHandled(), it should ret
urn an empty list!. Fix it!",
                            h.getClass().getCanonicalName());
                } else {
                    sb.append("[");
                    for (Class<?> c : h.contentHandled()) {
                        String sn = c.getSimpleName();
                        sb.append(sn).append(' ');
                    }
                    sb.append(']');
                }
            }
            sb.append("; ");
```

```
        }
        return sb.toString();
    }

    /**
     * Provides an {@code Handler} associated with the provided content
     *
     * @param content the {@code Message content}
     * @return an Optional that contains the {@code Handler} if one is found
     */
    Optional<Handler> getHandlerForContent(Object content) {
        return Optional.ofNullable(contentToHandlerMap.get(content.getClass()));
    }


    /**
     * Answers whether or not there is a {@code Handler} registered that can
     * process the specified content.
     *
     * @param contentClass the {@code Class} of the content
     * @return true if there is a handler, false otherwise.
     */
    public boolean canHandle(final Class<?> contentClass) {
        return contentToHandlerMap.containsKey(contentClass);
    }

    /**
     * Invokes all Agents at the current time.
     *
     * @param currentTime_ms the current simulation time
     * @return the result of the processing
     */
    public ProcessingResult runAllAgents(final long currentTime_ms) {
        checkInit();
        logger.trace("Entity {} entering runAllAgents", getOwnerName());
        List<ProcessingResult> output = new ArrayList<>();
        agents.forEach(agent -> {
            logger.trace("Entity {} invoking agent {} at {}",
                    getOwnerName(),
                    agent.getClass().getName(), currentTime_ms);
            ProcessingResult pr = agent.tick(currentTime_ms);
            logger.trace("In Entity {}, agent {} returned {}",
                    getOwnerName(),
                    agent.getClass().getName(),
                    pr);
            output.add(pr);
        });
        logger.trace("Entity {} runAllAgents returning {}", getOwnerName(), output);
        return ProcessingResult.merge(output);
    }

    private String getOwnerName() {
        return owner == null ? "NULL" : owner.getName();
    }

    /**
     * Initialized the set of {@code MessageHandlers} and {@code TickAgents}
     * <p>
     * This method injects For each handler and agent,
     */
    public void init() {
        if (owner == null) {
            throw new SSTAFException("Owner has not been set");
        }
        injectAll(owner);
        features.values().forEach(feature -> {
            if (feature instanceof Handler) {
                Handler handler = (Handler) feature;
```

```
                handler.contentHandled().forEach(message -> contentToHandlerMap.put(message, hand
ler));
                nameToHandlerMap.putIfAbsent(handler.getName(), handler);
            }
            if (feature instanceof Agent) {
                Agent agent = (Agent) feature;
                agents.add(agent);
            }
            feature.init();
        });
        initialized = true;
    }

    /**
     * Injects the provided {@code Object}s into the loaded {@code Provider}s
     *
     * @param items the items to inject
     */
    public void injectAll(final Object... items) {
        Injector.injectAll(this, items);
        features.values().forEach(provider -> Injector.injectAll(provider, items));
    }

    /**
     * Injects an Object into a named field
     *
     * @param attributeName  the name of the field
     * @param objectToInject the Object
     */
    public void injectNamed(final String attributeName, final Object objectToInject) {
        Injector.inject(this, attributeName, objectToInject);
        features.values().forEach(provider -> Injector.inject(provider, attributeName, objectToIn
ject));
    }

    /**
     * Registers a Handler
     *
     * @param handler the Handler
     * @param <T>     the Handler's type.
     */
    public <T extends Handler> void register(final T handler) {
        Objects.requireNonNull(handler, "Handler cannot be null");
        register((Feature) handler);
        handler.contentHandled().forEach(message -> contentToHandlerMap.put(message, handler));
        nameToHandlerMap.putIfAbsent(handler.getName(), handler);
    }

    /**
     * Registers an Agent
     *
     * @param agent the Agent
     * @param <T>   the Agent's type
     */
    public <T extends Agent> void register(final T agent) {
        Objects.requireNonNull(agent, "Agent cannot be null");
        register((Handler) agent);
        agents.add(agent);
    }

    /**
     * Registers a Feature
     *
     * @param feature the Feature
     * @param <T>     the type of the Feature
     */
    public <T extends Feature> void register(final T feature) {
        Objects.requireNonNull(feature, "Feature must not be null");
```

```
        FeatureSpecification ss = FeatureSpecification.builder()
                .featureClass(feature.getClass())
                .featureName(feature.getName())
                .majorVersion(feature.getMajorVersion())
                .minorVersion(feature.getMinorVersion())
                .build();
        features.putIfAbsent(ss, feature);
    }

    public Optional<FeatureSpecification> getSpecificationForHandler(final Class<?> contentClass)
{
        Handler handler = contentToHandlerMap.get(contentClass);
        if (handler == null) {
            return Optional.empty();
        } else {
            return Optional.of(FeatureSpecification.from(handler));
        }
    }
}
```

A.214 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/entity/Force.java

```
package mil.sstaf.core.entity;
public enum Force {
    SYSTEM,
    BLUE,
    RED,
    GRAY;
    }
```

A.215 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/entity/Human.java

```
package mil.sstaf.core.entity;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import com.fasterxml.jackson.databind.JsonNode;
import lombok.Getter;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import mil.sstaf.core.json.JsonLoader;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import java.io.File;
import java.nio.file.Path;
/**
 * Humans have multiple identifies. First, they have their definition name. This is the "definiti
onName" specified in
 * the JSON configuration. Multiple humans can share the same definition name. Second, there is t
he simple name. This
 * name identifies each instance of the Human. By default, it is a randomly-generated UUID.
 */
@SuperBuilder
@Jacksonized
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
public class Human extends BaseEntity {
    private static final Logger logger = LoggerFactory.getLogger(Human.class);
    @Getter
    private final String definitionName;
    protected Human(HumanBuilder<?, ?> builder) {
        super(builder);
        this.definitionName = builder.definitionName == null
                ? "UNSPECIFIED" : builder.definitionName;
    }
    @Override
    public String toString() {
        return "Human{" +
                ", definitionName='" + definitionName + '\'' +
                ", uuid=" + uuid +
                ", name='" + getName() + '\'' +
                "} " + super.toString();
    }
    /**
     * Performs any Human-level initialization
     */
    public void init() {
        logger.debug("Initializing Human {}", getPath());
        super.init();
    }
    public String getPath() {
        return Entity.ENTITY_PATH_DELIMITER + getName();
    }
    public static Human from(File file) {
        JsonLoader jsonLoader = new JsonLoader();
        return (Human) jsonLoader.load(Path.of(file.getPath()));
    }
    public static Human from(String json, Path sourceDir) {
        JsonLoader jsonLoader = new JsonLoader();
        return (Human) jsonLoader.load(json, sourceDir);
    }
    public static Human from(JsonNode jsonNode, Path sourceDir) {
        JsonLoader jsonLoader = new JsonLoader();
        return (Human) jsonLoader.load(jsonNode, sourceDir);
    }

}
```

A.216 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/entity/Message.java

```
package mil.sstaf.core.entity;
import mil.sstaf.core.features.HandlerContent;
/**
 * Defines Objects that can be sent between Entities
 */
public interface Message {
    /**
     * Provides the sequence number associated with the {@code Message}.
     *
     * @return the sequence number for the message
     */
    long getSequenceNumber();
    /**
     * Provides the contents of the message.
     *
     * @return the contents
     */
    HandlerContent getContent();
    /**
     * Provides the {@code Address} of the {@code Entity} and {@code Handler}
     * to which the {@code Message} must go.
     *
     * @return the destination {@code Address}
     */
    Address getDestination();
    /**
     * Provides the {@code Address} of the {@code Entity} and {@code Handler}
     * from which the {@code Message} originated.
     *
     * @return the source {@code Address}
     */
    Address getSource();
    /**
     * Provides the {@code Address} of the {@code Entity} and {@code Handler}
     * to which the destination {@code Entity} and {@code Handler} should send
     * the result of processes the content.
     *
     * @return the respondTo {@code Address}
     */
    Address getRespondTo();
}
```

A.217 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/entity/MessageDriven.java

```
package mil.sstaf.core.entity;
import java.util.List;
import java.util.concurrent.atomic.AtomicLong;
/**
 * Interface for all object's that communicate via asynchronous {@code Message} objects.
 * <p>
 * Each instance of {@code MessageDriven} objects must have a globally unique identification numb
er. This
 * ID number is used to facilitate a repeatable global total ordering of messages and events with
in the
 * system. The {@code MessageDriven.IDProvider.getID()} method can be used to provide the indenti
fication number.
 */
public interface MessageDriven {
    long CLIENT_APP_ID = -1776;
    /**
     * Adds a {@link Message} to this object's {@code inboundQueue}.
     *
     * @param message the message to add
     */
    void receive(Message message);
    /**
     * Takes all of the messages currently enqueued in this object's {@code inboundQueue}.
     *
     * @return a {@link java.util.List} of all of the inbound messages;
     */
    List<Message> takeInbound();
    /**
     * Takes all of the messages currently enqueued in this object's {@code outboundQueue}.
     *
     * @return a {@link java.util.List} of all of the outbound messages;
     */
    List<Message> takeOutbound();
    /**
     * Provides the unique ID number for this {@code MessageDriven} object.
     *
     * @return the ID number.
     */
    long getId();
    /**
     * Provides a message sequence number that is unique and ordered within the context of
     * this {@code MessageDriven} object.
     * <p>
     * The sequence number supports the global total order mechanism.
     *
     * @return the sequence number
     */
    long generateSequenceNumber();
    class BlockCounter {
        public static final long SYSTEM_BLOCK_BEGIN = 0L;
        public static final long USER_BLOCK_BEGIN = 10000L;
        public static BlockCounter systemCounter
                = new BlockCounter("System", SYSTEM_BLOCK_BEGIN, USER_BLOCK_BEGIN);
        public static BlockCounter userCounter
                = new BlockCounter("User", USER_BLOCK_BEGIN, Long.MAX_VALUE);
        private final AtomicLong counter;
        private final long upperValue;
        private final String name;
        private BlockCounter(final String name, final long lowerValue, final long upperValue) {
            this.name = name;
            this.counter = new AtomicLong(lowerValue);
            this.upperValue = upperValue;
```

```
        }
        public long getID() {
            if (counter.get() == upperValue - 1) {
                throw new IllegalStateException(name + " entity id counter has exceeded maximum s
upported entity count");
            }
            return counter.getAndIncrement();
        }
    }
}
```

A.218 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/entity/MessageQueueComparator.java

```
package mil.sstaf.core.entity;
import java.util.Comparator;
public class MessageQueueComparator implements Comparator<Message> {
    @Override
    public int compare(Message msg1, Message msg2) {
        if (msg1 instanceof EntityEvent && msg2 instanceof EntityEvent) {
            EntityEvent event1 = (EntityEvent) msg1;
            EntityEvent event2 = (EntityEvent) msg2;
            return compareEvents(event1, event2);
        } else if (msg1 instanceof EntityEvent) {
            return 1;
        } else if (msg2 instanceof EntityEvent) {
            return -1;
        } else {
            return compareMessages(msg1, msg2);
        }
    }
    private int compareEvents(EntityEvent event1, EntityEvent event2) {
        if (event1.getEventTime_ms() < event2.getEventTime_ms()) {
            return -1;
        } else if (event1.getEventTime_ms() > event2.getEventTime_ms()) {
            return 1;
        } else {
            return compareMessages(event1, event2);
        }
    }
    private int compareMessages(Message msg1, Message msg2) {
        if (msg1 == null) return 1;
        else if (msg2 == null) return -1;
        else {
            Address source1 = msg1.getSource();
            Address source2 = msg2.getSource();
            if (source1 != null && source1.equals(source2)) {
                return Long.compare(msg1.getSequenceNumber(), msg2.getSequenceNumber());
            } else {
                return Address.COMPARATOR.compare(source1, source2);
            }
        }
    }
}
```

A.219 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/entity/MessageResponse.java

```
package mil.sstaf.core.entity;
import lombok.Data;
import lombok.EqualsAndHashCode;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
@EqualsAndHashCode(callSuper = true)
@Data
@SuperBuilder
@Jacksonized
public class MessageResponse extends SimpleMessage {
    private final long messageID;
}
```

A.220 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/entity/PathToHandleMapper.java

```
package mil.sstaf.core.entity;
import java.util.Optional;
/**
 * @author Ron Bowers
 * @version 1.0
 * @since 1.0
 *
 * <p>Single-method interface for objects that map a {@code Path} to
 * an {@code EntityHandle}.</p>
 */
@FunctionalInterface
public interface PathToHandleMapper {
    /**
     * Resolves the {@code String} that an {@code Entity} path to an {@code EntityHandle}.
     *
     * @param path the path to map to an {@code}.
     * @return an {@code Optional} containing the {@code EntityHandle} if it was found
     */
    Optional<EntityHandle> invoke(String path);
}
```

A.221 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/entity/Rank.java

```
package mil.sstaf.core.entity;
import java.util.Objects;
public enum Rank {
    E1("PVT"),
    E2("PV2"),
    E3("PVC"),
    E4("CPL"),
    E5("SGT"),
    E6("SSG"),
    E7("SFC"),
    E8("MSG"),
    E9("SGM"),
    WO1("WO1"),
    WO2("CW2"),
    WO3("CW3"),
    WO4("CW4"),
    WO5("CW5"),
    O1("2LT"),
    O2("1LT"),
    O3("CPT"),
    O4("MAJ"),
    O5("LTC"),
    O6("COL"),
    O7("BG"),
    O8("MG"),
    O9("LTG"),
    O10("GEN");
    private final String code;
    Rank(String abbreviation) {
        this.code = Objects.requireNonNull(abbreviation);
    }
    public static Rank findMatch(String code) {
        for (Rank rank : values()) {
            if (rank.matches(code)) {
                return rank;
            }
        }
        return null;
    }
    public String getCode() {
        return code;
    }
    public boolean matches(String code) {
        return this.name().equals(code) || this.code.equals(code);
    }
}
```

A.222 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/entity/SimpleMessage.java

```
package mil.sstaf.core.entity;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.experimental.SuperBuilder;
import mil.sstaf.core.features.HandlerContent;
import java.util.concurrent.atomic.AtomicLong;
/**
 * Base implementation of {@code Message}
 */
@SuperBuilder
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
@EqualsAndHashCode
public abstract class SimpleMessage implements Message {
    protected final static AtomicLong counter = new AtomicLong();
    @Getter
    protected final long sequenceNumber;
    @Getter
    protected final Address destination;
    @Getter
    protected final Address source;
    @Getter
    protected final Address respondTo;
    @Getter
    protected final HandlerContent content;
    protected SimpleMessage(SimpleMessageBuilder<?,?> builder) {
        sequenceNumber = counter.getAndIncrement();
        this.destination = builder.destination;
        this.source = builder.source;
        this.respondTo = builder.respondTo;
        this.content = builder.content;
    }

}
```

A.223 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/entity/Soldier.java

```
package mil.sstaf.core.entity;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import com.fasterxml.jackson.databind.JsonNode;
import lombok.Builder;
import lombok.Getter;
import lombok.Setter;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import mil.sstaf.core.json.JsonLoader;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import java.io.File;
import java.nio.file.Path;
@SuperBuilder
@Jacksonized
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
public class Soldier extends Human {
    private static final Logger logger = LoggerFactory.getLogger(Soldier.class);
    @Builder.Default
    private Rank rankEnum = Rank.E1;
    private String rank;
    //
    // Identity
    //
    @Setter
    @Getter
    private String position; // My position within a Unit, used a key in Unit
    @Getter
    @Setter
    private Unit unit; // My unit
    private String path; // Cached path from top unit to me.
    //
    // Soldier-specific services
    //
    protected Soldier(SoldierBuilder<?, ?> builder) {
        super(builder);
        this.rank = builder.rank == null ? "E1" : builder.rank;
        this.rankEnum = Rank.findMatch(this.rank);
    }
    public static Soldier from(File file) {
        JsonLoader jsonLoader = new JsonLoader();
        return (Soldier) jsonLoader.load(Path.of(file.getPath()));
    }
    public static Soldier from(String json, Path sourceFile) {
        JsonLoader jsonLoader = new JsonLoader();
        return (Soldier) jsonLoader.load(json, sourceFile);
    }
    public static Soldier from(JsonNode jsonNode, Path sourceFile) {
        JsonLoader jsonLoader = new JsonLoader();
        return (Soldier) jsonLoader.load(jsonNode, sourceFile);
    }
    public String getPath() {
        if (path == null) {
            if (unit == null) {
                path = Entity.ENTITY_PATH_DELIMITER + super.getName();
            } else {
                path = unit.getPath() + Entity.ENTITY_PATH_DELIMITER + position;
            }
        }
        return path;
    }
    public void setForce(final Force force) {
```

```
        super.setForce(force);
        path = null;
    }
    public Rank getRank() {
        return rankEnum;
    }
    public void setUnit(final Unit unit, String position) {
        this.path = null;
        this.unit = unit;
        this.position = position;
    }
    /**
     * Performs any Soldier-level initialization
     */
    public void init() {
        logger.debug("Initializing Soldier {}", getPath());
        super.init();
    }
}
```

A.224 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/entity/Unit.java

```
package mil.sstaf.core.entity;
import com.fasterxml.jackson.annotation.JsonIgnore;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import com.fasterxml.jackson.databind.JsonNode;
import lombok.Builder;
import lombok.Getter;
import lombok.NonNull;
import lombok.Singular;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import mil.sstaf.core.json.JsonLoader;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import java.io.File;
import java.nio.file.Path;
import java.util.*;
@SuperBuilder
@Jacksonized
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
public class Unit extends BaseEntity {
    private static final Logger logger = LoggerFactory.getLogger(Unit.class);
    private static final String COMMANDER = "Commander";
    @Singular
    private final List<MemberSoldier> soldiers;
    @Singular
    private final List<MemberUnit> subUnits;
    private final UnitType type;
    @Getter
    @Builder.Default
    @JsonIgnore
    private Map<String, Soldier> soldierMap = null;
    @Builder.Default
    @JsonIgnore
    private Map<String, Unit> subUnitMap = null;
    @Builder.Default
    @JsonIgnore
    private String path = null;
    @Builder.Default
    @JsonIgnore
    private Unit parent = null;
    /**
     * Constructor
     *
     * @param builder the builder
     */
    protected Unit(UnitBuilder<?, ?> builder) {
        super(builder);
        type = builder.type;
        soldiers = builder.soldiers;
        subUnits = builder.subUnits;
        soldierMap = compileSoldiers(builder);
        subUnitMap = compileUnits(builder);
        for (var e : subUnitMap.entrySet()) {
            e.getValue().setName(e.getKey());
        }
        resetSoldiersUnit();
        resetSubUnitsParent();
    }
    public static Unit from(File file) {
        JsonLoader jsonLoader = new JsonLoader();
        return (Unit) jsonLoader.load(Path.of(file.getPath()));
    }
```

```
    public static Unit from(String json, Path sourceFile) {
        JsonLoader jsonLoader = new JsonLoader();
        return (Unit) jsonLoader.load(json, sourceFile);
    }
    public static Unit from(JsonNode jsonNode, Path sourceFile) {
        JsonLoader jsonLoader = new JsonLoader();
        return (Unit) jsonLoader.load(jsonNode, sourceFile);
    }
    private Map<String, Soldier> compileSoldiers(UnitBuilder<?, ?> builder) {
        Map<String, Soldier> soldierMap = new HashMap<>();
        if (builder.soldiers != null) {
            for (MemberSoldier sp : builder.soldiers) {
                String position = sp.getPosition();
                Soldier soldier = sp.getSoldier();
                soldier.setUnit(this, position);
                soldierMap.put(position, soldier);
            }
        }
        return soldierMap;
    }
    private Map<String, Unit> compileUnits(UnitBuilder<?, ?> builder) {
        Map<String, Unit> unitMap = new TreeMap<>();
        if (builder.subUnits != null) {
            for (MemberUnit mu : builder.subUnits) {
                Unit unit = mu.unit;
                String label = mu.label;
                unitMap.put(label, unit);
                unit.setName(label);
                unit.parent = this;
            }
        }
        return unitMap;
    }
    /**
     * Performs any Unit-level initialization
     */
    public void init() {
        logger.debug("Initializing Unit {}", getPath());
        super.init();
    }
    private void resetSubUnitsParent() {
        for (Map.Entry<String, Unit> entry : subUnitMap.entrySet()) {
            entry.getValue().attach(this, entry.getKey());
        }
    }
    public void setName(final String name) {
        this.name = name;
        this.path = null;
    }
    public String getPath() {
        if (path == null) {
            path = (parent == null) ? name : parent.getPath() + Entity.ENTITY_PATH_DELIMITER + na
me;
        }
        return path;
    }
    private void checkString(final String name) {
        Objects.requireNonNull(name, "Name must not be null");
    }
    private void checkSoldier(final Soldier s) {
        Objects.requireNonNull(s, "Soldier must not be null");
    }
    public Soldier getCommander() {
        return soldierMap.get(COMMANDER);
    }
    public void setCommanderByName(final String name) {
        checkString(name);
        Soldier newCommander = getSoldier(name);
```

```
        if (newCommander != null) {
            Soldier oldCommander = soldierMap.remove(COMMANDER);
            if (oldCommander != null) {
                addSoldier(newCommander.getPosition(), oldCommander);
            }
            addSoldier(COMMANDER, newCommander);
        }
    }
    private Soldier getSoldier(String name) {
        checkString(name);
        Soldier result = null;
        for (Soldier s : soldierMap.values()) {
            if (s.getName().equals(name)) {
                result = s;
                break;
            }
        }
        return result;
    }
    public void detach() {
        if (parent != null) {
            parent.subUnitMap.remove(name);
        }
        this.parent = null;
        this.path = null;
        resetSoldiersUnit();
        resetSubUnitsParent();
    }
    public void attach(final Unit parent, final String key) {
        checkUnit(parent);
        this.parent = parent;
        this.name = key;
        this.path = null;
        resetSoldiersUnit();
        resetSubUnitsParent();
    }
    private void resetSoldiersUnit() {
        for (Map.Entry<String, Soldier> entry : soldierMap.entrySet()) {
            entry.getValue().setUnit(this, entry.getKey());
        }
    }
    public UnitType getType() {
        return type;
    }
    public Soldier getSoldierByPosition(final String path) {
        checkString(path);
        int firstSeparator = path.indexOf(':');
        if (firstSeparator == 0) {
            //
            // path starts with ':' not sensible
            //
            return null;
        } else if (firstSeparator < 0) {
            //
            // Simple name
            //
            return soldierMap.get(path);
        } else {
            String firstToken = path.substring(0, firstSeparator);
            String remainder = path.substring(firstSeparator + 1);
            if (this.name.equals(firstToken)) {
                return getSoldierByPosition(remainder);
            } else {
                Unit subUnit = subUnitMap.get(firstToken);
                if (subUnit == null) {
                    return null;
                } else {
                    return subUnit.getSoldierByPosition(remainder);
```

```
                }
            }
        }
    }
    public Soldier getSoldierByName(final String name) {
        checkString(name);
        Soldier result = getSoldier(name);
        for (Iterator<Unit> iterator = subUnitMap.values().iterator();
             result == null && iterator.hasNext(); ) {
            Unit unit = iterator.next();
            result = unit.getSoldierByName(name);
        }
        return result;
    }
    public Soldier removeSoldierByName(final String name) {
        checkString(name);
        Soldier s = getSoldierByName(name);
        if (s != null) {
            Unit u = s.getUnit();
            String position = s.getPosition();
            if (u == null || position == null) {
                throw new IllegalStateException("A Soldier in a unit must have non-null Unit and
Position values");
            }
            s = u.soldierMap.remove(position);
            s.setUnit(null, null);
            return s;
        }
        return null;
    }
    public Map<String, Soldier> getAllMembers() {
        Map<String, Soldier> members = new TreeMap<>();
        for (Map.Entry<String, Soldier> entry : soldierMap.entrySet()) {
            members.put(entry.getValue().getPath(), entry.getValue());
        }
        for (Unit unit : subUnitMap.values()) {
            members.putAll(unit.getAllMembers());
        }
        return members;
    }
    public Map<String, Soldier> getDirectMembers() {
        Map<String, Soldier> members = new TreeMap<>();
        for (Map.Entry<String, Soldier> entry : soldierMap.entrySet()) {
            members.put(entry.getValue().getPath(), entry.getValue());
        }
        return members;
    }
    public Map<String, Unit> getSubUnitMap() {
        return Collections.unmodifiableMap(subUnitMap);
    }
    public Map<String, Unit> getAllUnits() {
        Map<String, Unit> units = new TreeMap<>();
        units.put(this.getPath(), this);
        for (Unit unit : subUnitMap.values()) {
            units.putAll(unit.getAllUnits());
        }
        return units;
    }
    public int getNumMembers() {
        int numMembers = soldierMap.size();
        for (Unit unit : subUnitMap.values()) {
            numMembers += unit.getNumMembers();
        }
        return numMembers;
    }
    String getPositionForSoldierByName(final String name) {
        checkString(name);
        Soldier s = getSoldierByName(name);
```

```
        return (s == null) ? null : s.getPosition();
    }
    void transferSoldierTo(final String name, final Unit receiver, final String position) {
        checkString(name);
        checkString(position);
        checkUnit(receiver);
        Soldier s = removeSoldierByName(name);
        if (s != null) {
            receiver.addSoldier(position, s);
        }
    }
    public void addSoldier(final Soldier soldier) {
        checkSoldier(soldier);
        addSoldier(soldier.getName(), soldier);
    }
    public void addSoldier(final String position, final Soldier soldier) {
        checkString(position);
        checkSoldier(soldier);
        soldierMap.putIfAbsent(position, soldier);
        soldier.setUnit(this, position);
    }
    public void addUnit(final String label, final Unit unit) {
        checkUnit(unit);
        subUnitMap.putIfAbsent(label, unit);
        unit.attach(this, unit.getName());
    }
    private void checkUnit(final Unit unit) {
        Objects.requireNonNull(unit, "Unit must not be null");
    }
    /**
     * Defines the job a Soldier has in a unit.
     */
    @Builder
    @Jacksonized
    static public final class MemberSoldier {
        @Getter
        @NonNull
        String position;
        @Getter
        @NonNull
        Soldier soldier;
        public static MemberSoldier of(String position, Soldier soldier) {
            Objects.requireNonNull(position);
            Objects.requireNonNull(soldier);
            return builder().position(position).soldier(soldier).build();
        }
    }
    @Builder
    @Jacksonized
    static public final class MemberUnit {
        @Getter
        @NonNull
        String label;
        @Getter
        @NonNull
        Unit unit;
        public static MemberUnit of(String label, Unit definition) {
            Objects.requireNonNull(label);
            Objects.requireNonNull(definition);
            return builder().label(label).unit(definition).build();
        }
    }
}
```

A.225 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/entity/UnitType.java

```
package mil.sstaf.core.entity;
public enum UnitType {
    FireTeam,
    Squad,
    Platoon,
    Company,
    Battalion,
    Brigade,
    Division
}
```

A.226 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/entity/package-info.java

```
/**
 * This package provides support for the Entity hierarchy. Entity instances
 * are used to represent various actors in SSTAF-based simulations.
 */
package mil.sstaf.core.entity;
```

A.227 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/features/Agent.java

```
package mil.sstaf.core.features;
/**
 * Interface for classes that should be triggered on every tick regardless
 * of whether or not a message was received.
 */
public interface Agent
        extends Handler {
    /**
     * Activate the Agent to perform a function at the specified time.
     *
     * @param currentTime_ms the simulation time.
     * @return a {@code ProcessingResult} containing internal and external {@code Message}s
     */
    ProcessingResult tick(final long currentTime_ms);
}
```

A.228 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/features/AppAdapter.java

```
package mil.sstaf.core.features;
import java.io.IOException;
import java.util.List;
/**
 * Base inteface for classes that support interaction with external applications.
 * <p>
 * Only one outbound and one inbound parameter is supported in the interface. I is expected that
multiple
 * parameters will be marshalled or bundled.
 */
public interface AppAdapter {
    /**
     * Activates the helper application and provides a {@link AppSession} object through which
     * the client can communicate. Uses the currently set arguments.
     *
     * @return a {@code Session}
     * @throws IOException if any of the file or process actions fail
     */
    AppSession activate() throws IOException;
    /**
     * Sets the arguments
     *
     * @param args the arguments to provide to the command.
     */
    void setArgs(List<String> args);
    /**
     * Activates the helper application using the specified arguments and provides a {@link AppSe
ssion} object
     * through which the client can communicate.
     *
     * @param args the arguments to provide to the application
     * @return a {@code Session}
     * @throws IOException if any of the file or process actions fail
     */
    AppSession activate(List<String> args) throws IOException;
    /**
     * Retrieves information about extracted resources
     * @return an {@code ExtractedResources} object
     */
    ResourceManager getResourceManager();

}
```

A.229 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/features/AppConfiguration.java

```
package mil.sstaf.core.features;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.Getter;
import lombok.Singular;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import java.util.List;
@SuperBuilder
@Jacksonized
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
public class AppConfiguration extends FeatureConfiguration {
    @Getter
    @Singular
    private List<String> resources;
    @Getter
    @Singular
    private List<String> processArgs;
    @Getter
    private AppSupport.Mode mode;
    @Getter
    private Class<? extends Feature> resourceOwner;
}
```

A.230 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/features/AppSession.java

```
package mil.sstaf.core.features;
import java.io.IOException;
/**
 * Base interface for interacting with an instance of an external helper application.
 */
public interface AppSession extends AutoCloseable{
    /**
     * Sends a message of type {@code P} to the helper and retrieves the response
     * @param cmd the command to send
     * @return the response from the helper
     * @throws IOException if communication fails
     */
    String invoke(String cmd) throws IOException;
}
```

A.231 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/features/AppSupport.java

```
package mil.sstaf.core.features;
import mil.sstaf.core.util.SSTAFException;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import java.io.*;
import java.util.List;
/**
 * Infrastructure for starting and interacting with external applications
 */
public class AppSupport {
    private static final Logger logger = LoggerFactory.getLogger(AppSupport.class);
    public static AppAdapter createAdapter(AppConfiguration configuration) {
        try {
            switch (configuration.getMode()) {
                case TRANSIENT:
                    return new TransientAdapter(configuration);
                case DURABLE:
                default:
                    return new DurableAdapter(configuration);
            }
        } catch (IOException ie) {
            throw new SSTAFException(ie);
        }
    }
    /**
     * Determines the working directory for the process.
     * <p>
     * To determine the working directory, the args are scanned to find one that matches the name
of
     * an extracted resource. The path of the extracted resource is then used as the working dire
ctory.
     *
     * @param args the args provided to start the process, one is assumed to be a file in the wor
king directory
     * @param rm   the {@code ResourceManager} object
     * @return a File that is the working directory
     */
    static File determineWorkDir(List<String> args, ResourceManager rm) {
        for (String arg : args) {
            for (var entry : rm.getResourceFiles().entrySet()) {
                String resourcePath = entry.getValue().getAbsolutePath();
                int idx = resourcePath.indexOf(arg);
                logger.debug("arg = {} resourcePath = {} idx = {}", arg, resourcePath, idx);
                if (idx > 0) {
                    String trimmed = resourcePath.substring(0, idx - 1);
                    logger.debug("trimmed = {}", trimmed);
                    File candidate = new File(trimmed);
                    if (candidate.exists() && candidate.canRead() && candidate.canWrite() && cand
idate.canExecute()) {
                        logger.debug("Working directory is: {}", candidate);
                        return candidate;
                    }
                }
            }
        }
        return rm.getDirectory().toFile();
    }
    /**
     * Enum for selecting whether applications are durable or transient.
     * Durable application stay open during
     */
    public enum Mode {
```

```
        TRANSIENT, DURABLE
    }

    static class TransientAdapter implements AppAdapter {
        Logger logger = LoggerFactory.getLogger(AppSupport.TransientAdapter.class);
        List<String> args;
        boolean dirtyArgs;
        ResourceManager resourceManager;
        TAppSession session;
        TransientAdapter(AppConfiguration configuration) throws IOException {
            resourceManager = ResourceManager.getManager(configuration.getResourceOwner());
            dirtyArgs = false;
            this.args = configuration.getProcessArgs();
        }
        @Override
        public AppSession activate() throws IOException {
            if (session != null) {
                session.close();
            }
            session = new TAppSession();
            return session;
        }
        @Override
        public AppSession activate(List<String> args) throws IOException {
            setArgs(args);
            return activate();
        }
        @Override
        public void setArgs(List<String> args) {
            this.args = args;
            dirtyArgs = true;
        }
        @Override
        public ResourceManager getResourceManager() {
            return resourceManager;
        }
        class TAppSession implements AppSession {
            Process process;
            BufferedReader input;
            BufferedWriter output;
            TAppSession() throws IOException {
                File workDir = determineWorkDir(args, resourceManager);
                logger.debug("workDir = {}", workDir);
                ProcessBuilder pb = new ProcessBuilder();
                pb.directory(workDir);
                pb.command(args);
                process = pb.start();
                input = new BufferedReader(new InputStreamReader(process.getInputStream()));
                output = new BufferedWriter(new OutputStreamWriter(process.getOutputStream()));
            }
            @Override
            public String invoke(String cmd) throws IOException {
                output.write(cmd);
                if (!cmd.endsWith("\n")) output.write("\n");
                output.flush();
                return input.readLine();
            }
            @Override
            public void close() {
                process.destroy();
                session = null;
                if (logger.isDebugEnabled()) {
                    logger.debug(process.info().toString());
                }
            }
        }
    }
    static class DurableAdapter implements AppAdapter {
```

```
        Logger logger = LoggerFactory.getLogger(AppSupport.DurableAdapter.class);
        List<String> args;
        boolean dirtyArgs;
        ResourceManager resourceManager;
        DAppSession session;
        DurableAdapter(AppConfiguration configuration) throws IOException {
            resourceManager = ResourceManager.getManager(configuration.getResourceOwner());
            this.args = configuration.getProcessArgs();
            dirtyArgs = false;
        }
        @Override
        public AppSession activate() throws IOException {
            if (dirtyArgs && session != null) {
                session.process.destroy();
                session = null;
            }
            if (session == null) {
                session = new DAppSession();
                dirtyArgs = false;
            }
            return session;
        }
        @Override
        public AppSession activate(List<String> args) throws IOException {
            setArgs(args);
            return activate();
        }
        @Override
        public void setArgs(List<String> args) {
            this.args = args;
            dirtyArgs = true;
        }
        @Override
        public ResourceManager getResourceManager() {
            return resourceManager;
        }
        class DAppSession implements AppSession {
            Process process;
            BufferedReader input;
            BufferedWriter output;
            DAppSession() throws IOException {
                File workDir = determineWorkDir(args, resourceManager);
                logger.debug("workDir = {}", workDir);
                ProcessBuilder pb = new ProcessBuilder();
                pb.command(args);
                pb.directory(workDir);
                process = pb.start();
                input = new BufferedReader(new InputStreamReader(process.getInputStream()));
                output = new BufferedWriter(new OutputStreamWriter(process.getOutputStream()));
            }
            @Override
            public String invoke(String cmd) throws IOException {
                output.write(cmd);
                if (!cmd.endsWith("\n")) output.write("\n");
                output.flush();
                return input.readLine();
            }
            @Override
            public void close() {
                if (logger.isDebugEnabled()) {
                    logger.debug(process.info().toString());
                }
            }
        }
    }
}
```

A.232 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/features/BaseAgent.java

```
package mil.sstaf.core.features;
/**
 * Base class for {@code Agent}s.
 */
public abstract class BaseAgent
        extends BaseHandler
        implements Agent {
    /**
     * Constructor for subclasses.
     * <p>
     * Note that concrete implementations must have a no-args constructor to be loadable as servi
ces
     *
     * @param featureName           the name of this {@code Feature}
     * @param majorVersion          the major version number
     * @param minorVersion          the minor version number
     * @param patchVersion          the patch version number
     * @param requiresConfiguration whether or not the Handler must be provided a configuration t
o make it valid.
     * @param description           a more verbose description of the {@code Agent}
     */
    protected BaseAgent(String featureName, int majorVersion, int minorVersion, int patchVersion,
                        boolean requiresConfiguration, String description) {
        super(featureName, majorVersion, minorVersion, patchVersion, requiresConfiguration, descr
iption);
    }
}
```

A.233 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/features/BaseFeature.java

```
package mil.sstaf.core.features;

import mil.sstaf.core.entity.EntityHandle;
import mil.sstaf.core.util.Injected;
import mil.sstaf.core.util.SSTAFException;
import mil.sstaf.core.util.Validation;
import org.slf4j.LoggerFactory;
import org.slf4j.Logger;

import java.util.Objects;

/**
 * Base class for implementation of {@code Feature}.
 */
public abstract class BaseFeature implements Feature {

    private static final Logger logger = LoggerFactory.getLogger(BaseFeature.class);

    protected final String featureName;
    protected final int majorVersion;
    protected final int minorVersion;
    protected final int patchVersion;
    protected final String description;
    protected final boolean requiresConfiguration;
    @Injected
    protected EntityHandle ownerHandle = null;
    protected boolean initialized = false;
    protected boolean configured = false;

    /**
     * Constructor for subclasses.
     * <p>
     * Note that concrete implementations must have a no-args constructor to be loadable as servi
ces
     *
     * @param featureName           the name of this {@code Feature}
     * @param majorVersion          the major version number
     * @param minorVersion          the minor version number
     * @param patchVersion          the patch version number
     * @param requiresConfiguration indicates whether a configuration must be provided for this F
eature to be valid
     * @param description           a more verbose description of the {@code Feature}
     */
    protected BaseFeature(String featureName, int majorVersion, int minorVersion, int patchVersio
n,
                          boolean requiresConfiguration, String description) {
        this.featureName = Objects.requireNonNull(featureName, "Feature name was null");
        this.description = Objects.requireNonNull(description, "Feature description was null");
        this.majorVersion = Validation.require(majorVersion, val -> val >= 0, "Major version was
< 0");
        this.minorVersion = Validation.require(minorVersion, val -> val >= 0, "Minor version was
< 0");
        this.patchVersion = Validation.require(patchVersion, val -> val >= 0, "Patch version was
< 0");
        this.requiresConfiguration = requiresConfiguration;
        logger.trace("Created {}", this);
    }

    /**
     * Provides a FeatureConfiguration
     * @return the default
     */
```

```
    @Override
    public Class<? extends FeatureConfiguration> getConfigurationClass() {
        return FeatureConfiguration.class;
    }

    /**
     * Provides the EntityHandle for the owning Entity
     *
     * @return the owner's EntityHandle
     */
    @Override
    public EntityHandle getOwner() {
        return ownerHandle;
    }

    /**
     * {@inheritDoc}
     */
    @Override
    public String toString() {
        return featureName + " "
                + majorVersion + "." + minorVersion + "." + patchVersion
                + " [" + description + "]";
    }

    /**
     * Provides the name of the service.
     *
     * @return the name of the service
     */
    @Override
    public String getName() {
        return featureName;
    }

    /**
     * Provides a description of the service.
     *
     * @return the description
     */
    @Override
    public String getDescription() {
        return description;
    }

    /**
     * Provides the major version of the service.
     *
     * @return the major version number
     */
    @Override
    public int getMajorVersion() {
        return majorVersion;
    }

    /**
     * Provides the minor version number of the service
     *
     * @return the minor version number.
     */
    @Override
    public int getMinorVersion() {
        return minorVersion;
    }

    /**
     * Provides the patch version number of the service
     *
```

```
     * @return the patch version number
     */
    @Override
    public int getPatchVersion() {
        return patchVersion;
    }

    /**
     * Initializes this {@code Feature}
     *
     * @throws SSTAFException if an error occurs.
     */
    @Override
    public void init() throws SSTAFException {
        if (requiresConfiguration && !configured) {
            String ownerString = ownerHandle == null ? "Unknown" : ownerHandle.getPath();
            throw new IllegalStateException(ownerString + ":" + featureName + " was not configure
d");
        }
        initialized = true;
        logger.trace("{}/BaseFeature initialized", featureName);
    }

    /**
     * Sets the configuration for this provider.
     * The configuration is applied when {@code init()} is invoked.
     *
     * @param configuration The configuration for the {@code Feature}
     */
    @Override
    public void configure(FeatureConfiguration configuration) {
        configured = true;
    }

    /**
     * Returns whether the Feature has been configured.
     * <p>
     * Note that subclasses must call up through super.configure() to make this true
     *
     * @return true if the Feature has been configured.
     */
    @Override
    public boolean isConfigured() {
        return configured;
    }

    /**
     * Returns whether the Feature has been initialized.
     * <p>
     * Note that subclasses must call up through super.init() to make this true
     *
     * @return true if the Feature has been initialized.
     */
    @Override
    public boolean isInitialized() {
        return initialized;
    }

}
```

A.234 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/features/BaseHandler.java

```
package mil.sstaf.core.features;
import mil.sstaf.core.entity.Address;
import mil.sstaf.core.entity.ErrorResponse;
import mil.sstaf.core.entity.Message;
import mil.sstaf.core.entity.MessageResponse;
import org.slf4j.LoggerFactory;
import org.slf4j.Logger;
import java.util.List;
import java.util.Objects;
/**
 * Base class for {@code Handler}s and {@code Agent}s.
 */
public abstract class BaseHandler
        extends BaseFeature
        implements Handler {
    private static final Logger logger = LoggerFactory.getLogger(BaseHandler.class);
    private Address address = null;
    /**
     * Constructor for subclasses.
     * <p>
     * Note that concrete implementations must have a no-args constructor to be loadable as servi
ces
     *
     * @param featureName           the name of this {@code Feature}
     * @param majorVersion          the major version number
     * @param minorVersion          the minor version number
     * @param patchVersion          the patch version number
     * @param requiresConfiguration whether or not the Handler must be provided a configuration t
o make it valid.
     * @param description           a more verbose description of the {@code Feature}
     */
    protected BaseHandler(String featureName, int majorVersion, int minorVersion, int patchVersio
n, boolean requiresConfiguration, String description) {
        super(featureName, majorVersion, minorVersion, patchVersion, requiresConfiguration, descr
iption);
    }
    @Override
    public void configure(FeatureConfiguration configuration) {
        super.configure(configuration);
    }
    /**
     * Provides the full Address for the Handler
     *
     * @return the Handler's Address
     */
    @Override
    public Address getAddress() {
        return address;
    }
    /**
     * Provides a descriptive String that fully identifies the Handler
     *
     * @return a descriptive String
     */
    @Override
    public String getInfoString() {
        String path = ownerHandle == null ? "null" : ownerHandle.getPath();
        String entityName = ownerHandle == null ? "null" : ownerHandle.getName();
        long id = ownerHandle == null ? -99999 : ownerHandle.getId();
        String handlerName = getName();
        return path + "[" + entityName + "|" + id + "][" + handlerName + "]";
    }
```

```
    /**
     * {@inheritDoc}
     */
    public void init() {
        super.init();
        if (ownerHandle == null) {
            throw new IllegalStateException("Owner handle has not been injected");
        } else {
            address = Address.makeAddress(ownerHandle, featureName);
        }
        logger.trace("{}/BaseHandler initialized", featureName);
    }
    /**
     * {@inheritDoc}
     */
    @Override
    public List<Class<? extends HandlerContent>> contentHandled() {
        return List.of();
    }
    /**
     * Creates a {@link ProcessingResult} that reports that the message is not supported.
     *
     * @param unsupported the message content class that is not supported by this {@code Handler}
     * @param sourceID    the id number of the original message
     * @param destination to where this message should go
     * @param exception   an exception that captures the stack trace from the handler
     * @return a {@code ProcessingResult}
     */
    protected ProcessingResult buildUnsupportedMessageResponse(final Object unsupported,
                                                               final long sourceID,
                                                               final Address destination,
                                                               final Throwable exception) {
        Objects.requireNonNull(unsupported, "unsupported");
        Objects.requireNonNull(destination, "destination");
        Objects.requireNonNull(exception, "exception");
        String desc = "Message class '" + unsupported.getClass() + "' is not supported.";
        var b = ErrorResponse.builder()
                .source(Address.makeAddress(this.ownerHandle, this.getName()))
                .destination(destination)
                .messageID(sourceID)
                .sequenceNumber(this.ownerHandle.getMessageSequenceNumber())
                .content(ExceptionCommand.builder().thrown(exception).build())
                .errorDescription(desc);
        Message out = b.build();
        logger.trace("Entity {} sending {}", ownerHandle.getName(), out);
        return ProcessingResult.of(out);
    }
    /**
     * Builds a Message to report an exception or error.
     *
     * @param message     the error message
     * @param exception   the exception
     * @param sourceID    the ID number of the message that caused the exception.
     * @param destination the {@code Entity} to which to send the message
     * @return a new Message
     */
    protected Message buildErrorResponse(final String message, final Throwable exception, final l
ong sourceID,
                                         final Address destination) {
        Objects.requireNonNull(destination, "Destination address must not be null");
        var b = ErrorResponse.builder()
                .source(Address.makeAddress(this.ownerHandle, this.getName()))
                .destination(destination)
                .messageID(sourceID)
                .sequenceNumber(this.ownerHandle.getMessageSequenceNumber())
                .content(ExceptionCommand.builder().thrown(exception).build())
                .errorDescription(message);
        Message out = b.build();
```

```
        logger.trace("Entity {} sending {}", ownerHandle.getName(), out);
        return out;
    }
    /**
     * Builds a Message to report the successful result of handling a message.
     *
     * @param response    the contents of the message
     * @param sourceID    the ID number of the message that caused the exception.
     * @param destination the {@code Entity} to which to send the message
     * @return a new Message
     */
    protected Message buildNormalResponse(HandlerContent response, final long sourceID, final Add
ress destination) {
        Objects.requireNonNull(destination, "Destination address must not be null");
        var b = MessageResponse.builder()
                .source(Address.makeAddress(this.ownerHandle, this.getName()))
                .destination(destination)
                .messageID(sourceID)
                .sequenceNumber(this.ownerHandle.getMessageSequenceNumber())
                .content(response);
        Message out = b.build();
        logger.trace("Entity {} sending {}", ownerHandle.getName(), out);
        return out;
    }
}
```

A.235 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/features/ExceptionCommand.java

```
package mil.sstaf.core.features;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
@SuperBuilder
@Jacksonized
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
@EqualsAndHashCode(callSuper = true)
public class ExceptionCommand extends HandlerContent {
    @Getter
    private final Throwable thrown;
}
```

A.236 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/features/ExtractedResources.java

```
package mil.sstaf.core.features;
import java.io.File;
import java.util.Collections;
import java.util.Map;
/**
 * Container for information about resources that have been extracted to the file systems.
 */
public class ExtractedResources {
    public final File tempDir;
    public final Map<String, File> resourceFiles;
    ExtractedResources(File tempDir, Map<String, File> resourceFiles) {
        this.tempDir = tempDir;
        this.resourceFiles = Collections.unmodifiableMap(resourceFiles);
    }
}
```

A.237 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/features/Feature.java

```
package mil.sstaf.core.features;

import mil.sstaf.core.entity.EntityHandle;
import mil.sstaf.core.util.SSTAFException;

/**
 * Base interface for all Providers
 */
public interface Feature {

    /**
     * Provides the name of the service.
     *
     * @return the name of the service
     */
    String getName();

    /**
     * Provides a description of the service.
     *
     * @return the description
     */
    String getDescription();

    /**
     * Provides the {@code Class} for the configuration object. This
     * is used to guide deserialization and ensure correct typing.
     * @return the {@code FeatureConfiguration}
     */
    Class<? extends FeatureConfiguration> getConfigurationClass();

    /**
     * Provides the major version of the service.
     *
     * @return the major version number
     */
    int getMajorVersion();

    /**
     * Provides the minor version number of the service
     *
     * @return the minor version number.
     */
    int getMinorVersion();

    /**
     * Provides the patch version number of the service
     *
     * @return the patch version number
     */
    int getPatchVersion();

    /**
     * Provides the EntityHandle for the owning Entity
     *
     * @return the owner's EntityHandle
     */
    EntityHandle getOwner();

    /**
     * Initializes this {@code Provider}
     *
```

```
     * @throws SSTAFException if an error occurs.
     */
    void init() throws SSTAFException;

    /**
     * Sets the configuration for this provider.
     * <p>
     * The configuration is applied when {@code init()} is invoked.
     */
    void configure(FeatureConfiguration configuration);

    /**
     * Returns whether the Feature has been configured.
     * <p>
     * Note that subclasses must call up through super.configure() to make this true
     *
     * @return true if the Feature has been configured.
     */
    boolean isConfigured();

    /**
     * Returns whether the Feature has been initialized.
     * <p>
     * Note that subclasses must call up through super.init() to make this true
     *
     * @return true if the Feature has been initialized.
     */
    boolean isInitialized();
}
```

A.238 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/features/FeatureConfiguration.java

```
package mil.sstaf.core.features;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.Getter;
import lombok.Setter;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import mil.sstaf.core.entity.Rank;
import java.util.Random;
/**
 * Base class for all configurations that can be applied to a {@code Feature.}
 *
 * The original approach was to pass in a {@code JSONObject} and a seed.
 * Converting to Lombok and Jackson made it easier to move deserialization into
 * the framework and provide the {@code Feature} with a real object.
 *
 * {@code Features}s are still responsible for validating their configuration
 * values.
 */
@Jacksonized
@SuperBuilder
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
public class FeatureConfiguration {
    @Getter
    @Setter
    private long seed;
    public FeatureConfiguration() {
        this.seed = new Random(System.currentTimeMillis() ^ System.nanoTime()).nextLong();
    }
}
```

A.239 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/features/FeatureLoader.java

```
package mil.sstaf.core.features;

import mil.sstaf.core.util.SSTAFException;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import java.lang.reflect.Constructor;
import java.lang.reflect.InvocationTargetException;
import java.util.*;
/**
 * Loads SSTAF features.
 *
 * <p>
 * SSTAF {@code Features} are dynamically loaded by the SSTAF core.
 * </p>
 */
final class FeatureLoader<F extends Feature> {
    private static final Logger logger = LoggerFactory.getLogger(FeatureLoader.class);
    private static final Set<Class<?>> registeredClasses = new HashSet<>();
    private final Class<F> type;
    private final List<Class<? extends F>> candidates;
    private final ModuleLayer moduleLayer;
    private final Object owner;
    /**
     * Instantiates a new {@code FeatureLoader}
     *
     * @param type  the type of the Feature
     * @param owner the owning object, used to determine the targeted Module.
     * @param layer the {@link ModuleLayer} into which to load the plugins.
     */
    FeatureLoader(final Class<F> type, Object owner, ModuleLayer layer) {
        this.type = type;
        this.moduleLayer = layer;
        this.owner = owner;
        //
        // Add a 'uses' relationship to the mil.sstaf.core module if necessary
        //
        this.getClass().getModule().addUses(type);
        if (owner != null) {
            owner.getClass().getModule().addUses(type);
        }
        candidates = new ArrayList<>(registeredClasses.size());
        for (Class<?> cls : registeredClasses) {
            if (type.isAssignableFrom(cls)) {
                candidates.add(cls.asSubclass(type));
            }
        }
    }
    /**
     * Registers a class with the SSTAF loading system. This enables classes to be
     * loaded and used without using ServiceLoader. This is useful in testing, but
     * might also be used to eliminate dynamic loading to enable use of GraalVM.
     *
     * @param cls the type to load
     */
    synchronized static void registerClass(Class<?> cls) {
        registeredClasses.add(cls);
    }
    /**
     * Clears the registered classes
     */
    synchronized static void clearClassRegistry() {
        registeredClasses.clear();
```

```
    }
    /**
     * Returns an unmodifiable view on the registered classes set
     *
     * @return the registered classes
     */
    synchronized static Set<Class<?>> getRegisteredClasses() {
        return Collections.unmodifiableSet(registeredClasses);
    }
    /**
     * Compares a {@code Feature} against a requirement
     *
     * @param candidate    the {@code Feature} object to be evaluated
     * @param modelName    the required model name
     * @param majorVersion the required major version number
     * @param minorVersion the required minor version number
     * @param exactVersion whether the version comparison requires an exact match or
     *                     the comparison will also accept newer versions.
     * @return whether the candidate meets the requirements
     */
    static <F extends Feature> boolean meetsRequirements(F candidate, final String modelName,
                                                         final int majorVersion,
                                                         final int minorVersion,
                                                         final boolean exactVersion) {
        if (candidate == null) {
            return false;
        }
        String candidateName = candidate.getName();
        int candidateMajorVersion = candidate.getMajorVersion();
        int candidateMinorVersion = candidate.getMinorVersion();
        boolean namesOK = modelName.equals("") ||
                (candidateName != null && candidateName.equals(modelName));
        boolean versionsEquals = candidateMajorVersion == majorVersion && candidateMinorVersion =
= minorVersion;
        boolean versionSufficient = candidateMajorVersion > majorVersion ||
                (candidateMajorVersion == majorVersion && candidateMinorVersion >= minorVersion);
        if (exactVersion) {
            return namesOK && versionsEquals;
        } else {
            return namesOK && versionSufficient;
        }
    }
    /**
     * Compares the version of the first Feature with the version of the second Feature
     * <p>
     * The behavior is that the value returned is negative if the first argument is "better"
     * and positive if the second is "better." Since better means higher version numbers,
     * regular numeric comparisons must be flipped.
     *
     * @param first  the first Feature
     * @param second the second Feature
     * @return a negative value if the first Feature is newer, a positive value if the second
     * Feature is newer, zero if they are equivalent.
     */
    static <F extends Feature> int compareVersions(final F first, final F second) {
        int major = -(Integer.compare(first.getMajorVersion(), second.getMajorVersion()));
        if (major == 0) {
            int minor = -(Integer.compare(first.getMinorVersion(), second.getMinorVersion()));
            if (minor == 0) {
                return -(Integer.compare(first.getPatchVersion(), second.getPatchVersion()));
            } else {
                return minor;
            }
        } else {
            return major;
        }
    }
    /**
```

```
     * Traverses an {@code Iterable} to find the {@code Feature} that best matches the
     * specification.
     *
     * @param candidates   the {@code Iterable} of Features to evaluate
     * @param modelName    the name of the Feature provided by its getName() method
     * @param majorVersion the major version of the Feature provided by its getMajorVersion() met
hod
     * @param minorVersion the minor version of the Feature provided by its getMinorVersion metho
d
     * @param exactVersion states whether the version of the found service must exactly match the
specified version
     * @return the selected Feature or null if none met the requirements.
     */
    static <F extends Feature> F findBestMatch(Iterable<F> candidates, F bestSoFar,
                                               final String modelName,
                                               final int majorVersion, final int minorVersion,
                                               final boolean exactVersion) {
        F selected = bestSoFar;
        for (F candidate : candidates) {
            if (meetsRequirements(candidate, modelName, majorVersion, minorVersion, exactVersion)
) {
                if (exactVersion) {
                    return candidate;
                } else if (selected == null) {
                    selected = candidate;
                    continue;
                }
                selected = compareVersions(selected, candidate) <= 0 ? selected : candidate;
            }
        }
        return selected;
    }
    /**
     * Loads a {@code Feature} using the specified parameters.
     * <p>
     * Method executes a series of strategies for loading the desired Feature. The best
     * matching feature, if any, is returned.
     *
     * @param modelName    the name of the service provided by its getName() method
     * @param majorVersion the major version of the service provided by its getMajorVersion() met
hod
     * @param minorVersion the minor version of the service provided by its getMinorVersion metho
d
     * @param exactVersion whether the version of the found service must exactly match the specif
ied version
     * @return a reference to the loaded object
     */
    F loadRef(final String modelName,
              final int majorVersion,
              final int minorVersion,
              final boolean exactVersion) {
        try {
            List<F> instances = generateFeatureInstances();
            F bestYet = findBestMatch(instances, null, modelName, majorVersion,
                    minorVersion, exactVersion);
            ServiceLoader<F> loader = ServiceLoader.load(moduleLayer, type);
            bestYet = findBestMatch(loader, bestYet, modelName, majorVersion,
                    minorVersion, exactVersion);
//            loader = ServiceLoader.load(type);
//            bestYet = findBestMatch(loader, bestYet, modelName, majorVersion,
//                      minorVersion, exactVersion);
            loader = ServiceLoader.load(type, Thread.currentThread().getContextClassLoader());
            bestYet = findBestMatch(loader, bestYet, modelName, majorVersion,
                    minorVersion, exactVersion);
            loader = ServiceLoader.loadInstalled(type);
            bestYet = findBestMatch(loader, bestYet, modelName, majorVersion,
                    minorVersion, exactVersion);
            return bestYet;
```

```
        } catch (InstantiationException |
                InvocationTargetException |
                NoSuchMethodException |
                IllegalAccessException |
                ServiceConfigurationError e) {
            String name;
            if (this.owner instanceof Feature) {
                name = ((Feature)owner).getName()+" (" + owner.getClass().getName() + ")";
            } else {
                name = owner.getClass().getName();
            }
            logger.error("In {}, could not load service {}, {}, {}", name, modelName, majorVersio
n, minorVersion);
            logger.error("{}", e.getMessage());
            e.printStackTrace(System.err);
            throw new SSTAFException(e);
        }
    }
    /**
     * Loads a {@code Feature} using the specified parameters.
     * <p>
     * Method executes a series of strategies for loading the desired Feature. The best
     * matching feature, if any, is returned.
     *
     * @param modelName    the name of the service provided by its getName() method
     * @param majorVersion the major version of the service provided by its getMajorVersion() met
hod
     * @param minorVersion the minor version of the service provided by its getMinorVersion metho
d
     * @param exactVersion whether the version of the found service must exactly match the specif
ied version
     * @return an {@code Optional} that will contains the service.
     */
    Optional<F> load(final String modelName,
                     final int majorVersion,
                     final int minorVersion,
                     final boolean exactVersion) {
        return Optional.ofNullable(loadRef(modelName, majorVersion, minorVersion, exactVersion));
    }
    /**
     * Creates instances of each candidate class so that the properties of the Feature
     * can be evaluated for suitability
     *
     * @return a {@code List} containing Features instantiated for each oif the candidate classes
     * @throws NoSuchMethodException     if a Class does not have a no-arg construction
     * @throws IllegalAccessException    if permissions are wrong
     * @throws InvocationTargetException if reflection goes wrong
     * @throws InstantiationException    if newInstance doesn't work
     */
    List<F> generateFeatureInstances() throws NoSuchMethodException,
            IllegalAccessException, InvocationTargetException, InstantiationException {
        List<F> instances = new ArrayList<>(candidates.size());
        for (Class<? extends F> cls : candidates) {
            Constructor<? extends F> constructor = cls.getConstructor();
            F instance = constructor.newInstance();
            instances.add(instance);
        }
        return instances;
    }
}
```

A.240 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/features/FeatureSpecification.java

```
package mil.sstaf.core.features;
import com.fasterxml.jackson.databind.JsonNode;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import mil.sstaf.core.json.JsonLoader;
import mil.sstaf.core.util.SSTAFException;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import java.io.File;
import java.nio.file.Path;
/**
 * Describes a Feature.
 * <p>
 * Used to specify which Feature to use.
 */
@SuperBuilder
@Jacksonized
public final class FeatureSpecification {
    private static final Logger logger = LoggerFactory.getLogger(FeatureSpecification.class);
    public final Class<? extends Feature> featureClass;
    public final String featureName;
    public final int majorVersion;
    public final int minorVersion;
    public final boolean requireExact;
    private FeatureSpecification(FeatureSpecificationBuilder<?, ?> builder) {
        if (builder.featureName == null) {
            throw new SSTAFException("featureName was null in FeatureSpecification");
        }
        this.featureClass = builder.featureClass;
        this.featureName = builder.featureName;
        this.majorVersion = builder.majorVersion;
        this.minorVersion = builder.minorVersion;
        this.requireExact = builder.requireExact;
    }
    /**
     * Creates a {@code FeatureSpecification} from a requirement.
     *
     * @param requires      the {@code Requires} annotation associated with the target field
     * @param providerClass the type of the target field.
     * @return a new {@code FeatureSpecification} matching the requirement specified by the Requi
res
     */
    public static FeatureSpecification from(final Requires requires, Class<? extends Feature> pro
viderClass) {
        return builder().featureName(requires.name())
                .majorVersion(requires.majorVersion())
                .minorVersion(requires.minorVersion())
                .featureClass(providerClass)
                .build();
    }
    /**
     * Creates a {@code FeatureSpecification} from an instantiated {@code Feature}
     *
     * @param f the {@code Feature}
     * @return a new {@code FeatureSpecification}
     */
    public static FeatureSpecification from(Feature f) {
        return builder().featureClass(f.getClass())
                .featureName(f.getName())
                .majorVersion(f.getMajorVersion())
                .minorVersion(f.getMinorVersion())
                .build();
```

```
    }
    /**
     * Answers whether the supplied specification meets the the specification required by this
     * {@code FeatureSpecification}
     *
     * @param candidate the {@code FeatureSpecification} to test
     * @return true if is satisfies the requirement, false otherwise.
     */
    public boolean isSatisfiedBy(final FeatureSpecification candidate) {
        boolean namesOK;
        boolean classesOK;
        boolean versionsEqual;
        boolean versionSufficient;
        if (logger.isTraceEnabled()) {
            logger.trace("Checking if {} satisfies {}", candidate, this);
        }
        if (candidate == null) {
            return false;
        } else if (this.equals(candidate)) {
            return true;
        } else {
            classesOK = featureClass == null || featureClass.isAssignableFrom(candidate.featureCl
ass);
            namesOK = featureName.isEmpty() || featureName.equals(candidate.featureName);
            versionsEqual = majorVersion == candidate.majorVersion && minorVersion == candidate.m
inorVersion;
            versionSufficient = candidate.majorVersion > majorVersion ||
                    (candidate.majorVersion == majorVersion && candidate.minorVersion >= minorVer
sion);
        }
        boolean answer = classesOK && namesOK && (this.requireExact ? versionsEqual : versionSuff
icient);
        if (logger.isTraceEnabled()) {
            logger.trace("classesOk = {} namesOK = {} versionsEqual = {} versionSufficient = {} r
equireExact = {} answer = {}",
                    classesOK, namesOK, versionsEqual, versionSufficient, requireExact, answer);
        }
        return answer;
    }
    /**
     * {@inheritDoc}
     */
    @Override
    public String toString() {
        return (featureClass == null ? "" : "(class " + featureClass.getSimpleName() + ") ")
                + featureName + " " + majorVersion + "." + minorVersion + "."
                + (requireExact ? " [EXACT]" : "");
    }
    public static FeatureSpecification from(File file) {
        JsonLoader jsonLoader = new JsonLoader();
        return jsonLoader.load(Path.of(file.getPath()),
                FeatureSpecification.class);
    }
    public static FeatureSpecification from(String json, Path sourceDir) {
        JsonLoader jsonLoader = new JsonLoader();
        return jsonLoader.load(json, FeatureSpecification.class, sourceDir);
    }
    public static FeatureSpecification from(JsonNode jsonNode, Path sourceDir) {
        JsonLoader jsonLoader = new JsonLoader();
        return jsonLoader.load(jsonNode, FeatureSpecification.class, sourceDir);
    }
}
```

A.241 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/features/Handler.java

```
package mil.sstaf.core.features;

import mil.sstaf.core.entity.Address;

import java.util.List;

/**
 * Specifies the interface for all objects that handle message content.
 */
public interface Handler
        extends Feature {

    /**
     * Provides the full Address for the Handler
     *
     * @return the Handler's Address
     */
    Address getAddress();

    /**
     * Provides a descriptive String that fully identifies the Handler
     *
     * @return a descriptive String
     */
    String getInfoString();

    /**
     * Provides a {@code List} of all of the message content {@code Class}es to which this
     * {@code Handle} responds.
     * <p>
     * Handlers can respond to multiple message classes. Only one Handler may be associated with
     * a message class.
     *
     * @return a {@code List} of {@code Class} objects.
     */
    List<Class<? extends HandlerContent>> contentHandled();

    /**
     * Initializes a {@code Handler}.
     * <p>
     * Initialization is invoked after configuration and dependency injection.
     * Init should check that all required dependencies are in place.
     */
    void init();

    /**
     * Performs the processing associated with the given argument at the specified time.
     * <p>
     * Both the scheduled time and current simulation time are reported. For an {@code EntityEven
t}
     * the scheduled time value will be less than or equal the current simulation time. For an
     * {@code EntityAction} scheduled time will be the same as the current time. Providing both t
imes
     * enables the Handler to adjust for any time difference between when an event was intended t
o occur
     * and when it was processed.
     *
     * @param arg              the message content
     * @param scheduledTime_ms the time for which an event was scheduled.
     * @param currentTime_ms   the current simulation time
     * @param from             the sources of the {@code Message}
     * @param id               the message sequence numner
```

```
     * @param respondTo        to where to send the results.
     * @return a {@code ProcessingResult} that contains the results of the processing
     */
    ProcessingResult process(HandlerContent arg, long scheduledTime_ms, long currentTime_ms,
                             Address from, long id, Address respondTo);


    /**
     * Defines a class that processes {@code Message} content.
     */
    interface ContentProcessor {
        ProcessingResult processMessageContent(HandlerContent content, long scheduledTime_ms, lon
g currentTime_ms, Address from, long id, Address respondTo);
    }

}
```

A.242 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/features/HandlerContent.java

```
package mil.sstaf.core.features;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.EqualsAndHashCode;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
@SuperBuilder
@Jacksonized
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
@EqualsAndHashCode
public class HandlerContent {
    protected HandlerContent() {    }
}
```

A.243 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/features/IntContent.java

```
package mil.sstaf.core.features;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
@SuperBuilder
@Jacksonized
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
@EqualsAndHashCode(callSuper = true)
public class IntContent extends HandlerContent {
    @Getter
    private final int intValue;
}
```

A.244 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/features/Loaders.java

```
package mil.sstaf.core.features;

import mil.sstaf.core.configuration.SSTAFConfiguration;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.lang.module.ModuleDescriptor;
import java.util.*;

/**
 * Static utilities for loading {@code Feature} services.
 */
public class Loaders {

    public static final String PRELOAD_PROPERTY = "sstaf.preloadFeatureClasses";
    private static final Logger logger = LoggerFactory.getLogger(Loaders.class);

    /**
     * Creates a new {@code FeatureLoader} parameterized to the required class type.
     *
     * @param type the {@code Class} of the desired service
     * @param <F>  the generic type
     * @return a new FeatureLoader
     */
    private static <F extends Feature> FeatureLoader<F> getFeatureLoader(Class<F> type, Object ow
ner, ModuleLayer moduleLayer) {
        return new FeatureLoader<>(type, owner, moduleLayer);
    }

    /**
     * Creates a new {@code FeatureLoader} parameterized to the required class type.
     *
     * @param type the {@code Class} of the desired service
     * @param <F>  the generic type
     * @return a new FeatureLoader
     */
    private static <F extends Feature> FeatureLoader<F> getFeatureLoader(Class<F> type) {
        return new FeatureLoader<>(type, null, SSTAFConfiguration.getInstance().getRootLayer());
    }

    /**
     * Registers a class with the SSTAF loading system. This enables classes to be
     * loaded and used without using ServiceLoader. This is useful in testing, but
     * might also be used to eliminate dynamic loading to enable use of GraalVM.
     *
     * @param cls the type to load
     */
    public static void registerClass(Class<?> cls) {
        FeatureLoader.registerClass(cls);
    }

    /**
     * Loads a {@code Feature} using the specified {@code ServiceSpecification}
     *
     * @param <T>   the type of the service
     * @param clazz the {@code Class} of the service
     * @param spec  the {@code ServiceSpecification} that specifies which service is required
     * @return an {@code Optional} that will contain the service if a match was found.
     */
    public static <T extends Feature> Optional<T> load(final Class<T> clazz, final FeatureSpecifi
cation spec) {
```

```
        return getFeatureLoader(clazz).load(spec.featureName, spec.majorVersion, spec.minorVersio
n, spec.requireExact);
    }

    /**
     * Loads a {@code Feature} using the specified {@code ServiceSpecification} into a specific
     * {@link ModuleLayer}.
     *
     * @param <T>         the type of the service
     * @param clazz       the {@code Class} of the service
     * @param spec        the {@code ServiceSpecification} that specifies which service is requir
ed
     * @param moduleLayer the {@code ModuleLayer} into which to load the {@code Feature}.
     * @return an {@code Optional} that will contain the service if a match was found.
     */
    public static <T extends Feature> Optional<T> load(final Class<T> clazz, final FeatureSpecifi
cation spec, Object owner, ModuleLayer moduleLayer) {
        return getFeatureLoader(clazz, owner, moduleLayer).load(spec.featureName, spec.majorVersi
on, spec.minorVersion, spec.requireExact);
    }

    /**
     * Loads a {@code Feature} using the specified {@code ServiceSpecification}
     *
     * @param <T>          the specific type of the {@code Feature}
     * @param clazz        the {@code Class} of the service
     * @param modelName    the name of the service provided by its getName() method
     * @param majorVersion the major version of the service provided by its getMajorVersion() met
hod
     * @param minorVersion the minor version of the service provided by its getMinorVersion metho
d
     * @param requireExact whether the version of the found service must exactly match the specif
ied version
     * @param moduleLayer  the {@link ModuleLayer} to use
     * @return the loaded Object or null if it could not be found.
     */
    public static <T extends Feature> T loadAsRef(final Class<T> clazz, final String modelName,
                                                  final int majorVersion, final int minorVersion,
                                                  boolean requireExact, ModuleLayer moduleLayer)
{
        return getFeatureLoader(clazz, null, moduleLayer).loadRef(modelName, majorVersion, minorV
ersion, requireExact);
    }

    /**
     * Loads a {@code Feature} using the specified parameters, assumes that an exact match is not
required.
     * <p>
     * Since an exact match is not required, any service with a matching name and a major and min
or version pair
     * greater than the specified version will be used. The behavior when multiple matching servi
ces are found
     * on the module path is not defined.
     *
     * @param <T>          the type of the service
     * @param clazz        the {@code Class} of the service
     * @param modelName    the name of the service provided by its getName() method
     * @param majorVersion the major version of the service provided by its getMajorVersion() met
hod
     * @param minorVersion the minor version of the service provided by its getMinorVersion metho
d
     * @return an {@code Optional} that contains the matching Feature.
     */
    public static <T extends Feature> Optional<T> load(final Class<T> clazz, final String modelNa
me,
                                                       final int majorVersion, final int minorVer
sion) {
        return getFeatureLoader(clazz).load(modelName, majorVersion, minorVersion, false);
```

```
    }

    /**
     * Loads a {@code Feature} using the specified parameters, assumes that an exact match is not
required.
     * <p>
     * Since an exact match is not required, any service with a matching name and a major and min
or version pair
     * greater than the specified version will be used. The behavior when multiple matching servi
ces are found
     * on the module path is not defined.
     *
     * @param <T>          the type of the service
     * @param clazz        the {@code Class} of the service
     * @param modelName    the name of the service provided by its getName() method
     * @param majorVersion the major version of the service provided by its getMajorVersion() met
hod
     * @param minorVersion the minor version of the service provided by its getMinorVersion metho
d
     * @param moduleLayer  the {@link ModuleLayer} to use
     * @return an {@code Optional} that contains the matching Feature.
     */
    public static <T extends Feature> Optional<T> load(final Class<T> clazz, final String modelNa
me,
                                                       final int majorVersion, final int minorVer
sion, ModuleLayer moduleLayer) {
        return getFeatureLoader(clazz, null, moduleLayer).load(modelName, majorVersion, minorVers
ion, false);
    }

    /**
     * Loads a {@code Feature} using the specified parameters.
     *
     * @param <T>          the type of the service
     * @param clazz        the {@code Class} of the service
     * @param modelName    the name of the service provided by its getName() method
     * @param majorVersion the major version of the service provided by its getMajorVersion() met
hod
     * @param minorVersion the minor version of the service provided by its getMinorVersion metho
d
     * @param exactVersion whether the version of the found service must exactly match the specif
ied version
     * @param moduleLayer  the {@code ModuleLayer} into which to load the {@code Feature}
     * @return an {@code Optional} that will contain the service if a match was found.
     */
    public static <T extends Feature> Optional<T> load(final Class<T> clazz, final String modelNa
me,
                                                       final int majorVersion, final int minorVer
sion,
                                                       final boolean exactVersion, ModuleLayer mo
duleLayer) {
        return getFeatureLoader(clazz, null, moduleLayer).load(modelName, majorVersion, minorVers
ion, exactVersion);
    }

    /**
     * Utility for diagnosing service problems.
     *
     * @param fs          the {@code FeatureSpecification}
     * @param featureType the type of the {@code Feature}
     * @param <T>         the type of the service
     * @return a descriptive {@code String}
     */
    public static <T extends Feature> String getHelpWithServices(FeatureSpecification fs, Class<T
> featureType) {

        return "SSTAF failed to load the specified service: '" + fs.toString()
                + " as an implementation of '" +
```

```
                featureType.getName() + "'\n" +
                "Check: \n" +
                "1. The jar containing the service is on the module path or the \n" +
                "   module and and its module path are specified in the SSTAFConfiguration\n" +
                "   or the Entity configuration file in which it is referenced.\n" +
                "2. The module-info specifies:\n   'provides " + featureType.getName() +
                " with NameOfTheImplementationClass'\n" +
                "3. The module-info specifies that it is open to mil.sstaf.core\n\n" +
                generateServiceReport(featureType);
    }


    /**
     * Prints all services matching the specified service type.
     *
     * @param serviceType the service to match
     * @param <T>         the service type.
     */
    public static <T extends Feature> String generateServiceReport(Class<T> serviceType) {
        StringBuilder sb = new StringBuilder("Found implementations of '").append(serviceType.get
Name()).append("'\n");
        ServiceLoader<T> sl = ServiceLoader.load(serviceType);
        for (Feature o : sl) {
            sb.append(String.format("%s %d.%d.%d\n",
                    o.getName(), o.getMajorVersion(),
                    o.getMinorVersion(), o.getPatchVersion()));
            Class<? extends Feature> cl = o.getClass();
            sb.append(String.format("    Implemented by: %s\n", cl.getName()));
            Module module = cl.getModule();
            sb.append(String.format("    Found in Module: %s\n", module.isNamed() ? module.getNam
e() : "UNNAMED"));

            ModuleDescriptor desc = module.getDescriptor();

            sb.append("    Module exports:\n");
            for (ModuleDescriptor.Exports exp : desc.exports()) {
                sb.append("        ").append(exp.toString()).append("\n");
            }

            sb.append("    Module provides:\n");
            for (ModuleDescriptor.Provides provides : desc.provides()) {
                sb.append("        ").append(provides.toString()).append("\n");
            }

            sb.append("    Module requires:\n");
            for (ModuleDescriptor.Requires requires : desc.requires()) {
                sb.append("        ").append(requires.toString()).append("\n");
            }

        }
        return sb.toString();
    }

    /**
     * Instantiates and registers a Class using its name.
     *
     * @param className the name of the Class.
     * @throws ClassNotFoundException if the class is not found in the classpath.
     */
    public static void registerClass(final String className) throws ClassNotFoundException {
        registerClass(Class.forName(className));
    }

    /**
     * Preloads 1 or more classes by name
     *
     * @param classes a varargs array of class names
     * @throws ClassNotFoundException if the class was not found in the classpath
```

```
     */
    public static void preloadFeatureClasses(List<String> classes) throws ClassNotFoundException
{
        Objects.requireNonNull(classes);
        if (Boolean.getBoolean(PRELOAD_PROPERTY)) {
            logger.debug("Preloading {} feature classes", classes.size());
            for (String s : classes) { // Using a loop rather than a lambda because of the except
ion
                logger.trace("Preloading {}", s);
                registerClass(s);
            }
        } else {
            logger.debug("Preloading disabled");
        }

    }

    /**
     * Preloads 1 or more classes by name
     *
     * @param classes a varargs array of class names
     * @throws ClassNotFoundException if the class was not found in the classpath
     */
    public static void preloadFeatureClasses(String... classes) throws ClassNotFoundException {
        Objects.requireNonNull(classes);
        preloadFeatureClasses(Arrays.asList(classes));
    }
}
```

A.245 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/features/LongContent.java

```
package mil.sstaf.core.features;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
@SuperBuilder
@Jacksonized
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
@EqualsAndHashCode(callSuper = true)
public class LongContent extends HandlerContent {
    @Getter
    private final long longValue;
}
```

A.246 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/features/ProcessingResult.java

```
package mil.sstaf.core.features;
import mil.sstaf.core.entity.EntityEvent;
import mil.sstaf.core.entity.Message;
import java.util.ArrayList;
import java.util.List;
/**
 * The result generated by {@code Handler}s and {@code Agent}s
 */
public class ProcessingResult {
    private static final ProcessingResult EMPTY_RESULT = new ProcessingResult();
    public final long nextEventTime_ms;
    public final List<Message> messages;
    /**
     * Private constructor
     *
     * @param messages a list of {@code Message}s for distribution to other {@code Entity} object
s,
     */
    private ProcessingResult(List<Message> messages) {
        this.messages = messages;
        long mt = Long.MAX_VALUE;
        for (Message msg : messages) {
            if (msg instanceof EntityEvent) {
                EntityEvent event = (EntityEvent) msg;
                mt = Long.min(mt, event.getEventTime_ms());
            }
        }
        nextEventTime_ms = mt;
    }
    /**
     * Private constructor for an empty {@code ProcessingResult}
     */
    private ProcessingResult() {
        this.messages = List.of();
        nextEventTime_ms = Long.MAX_VALUE;
    }
    /**
     * Factory method for making a ProcessingResult
     *
     * @param messages a list of {@code Message}s for distribution to this or other {@code Entity
} objects.
     * @return a new ProcessingResult
     */
    public static ProcessingResult of(final List<Message> messages) {
        return new ProcessingResult(messages);
    }
    /**
     * Factory method for making a ProcessingResult for a single {@code Message}
     *
     * @param message a single {@code Message} to be distributed to this or another {@code Entity
} object.
     * @return a new ProcessingResult
     */
    public static ProcessingResult of(final Message message) {
        return new ProcessingResult(List.of(message));
    }
    /**
     * Merge multiple {@code ProcessingResult}s into a single result.
     *
     * @param results the {@code ProcessingReuslt}s to merge
     * @return a new unified {@code ProcessingResult}
     */
```

```
    public static ProcessingResult merge(final List<ProcessingResult> results) {
        List<Message> accumulator = new ArrayList<>(results.size());
        results.forEach(pr -> {
            if (pr != null) {
                accumulator.addAll(pr.messages);
            }
        });
        return ProcessingResult.of(accumulator);
    }
    /**
     * Creates an empty {@code ProcessingResult}
     *
     * @return a new empty {@code ProcessingResult}
     */
    public static ProcessingResult empty() {
        return EMPTY_RESULT;
    }
}
```

A.247 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/features/Requires.java

```
package mil.sstaf.core.features;
import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;
/**
 * Annotation for specifying a dependency that must be injected.
 * <p>
 * The annotation can specify the name, major version, minor version and exact match
 * requirement for the Service to be injected.
 */
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.FIELD)
public @interface Requires {
    /**
     * Provides for a name or label to be associated with an injection point. This allows for mul
tiple
     * injection points of the same type to be differentiated.
     *
     * @return the name of the field.
     */
    String name() default "";
    /**
     * Specifies the major version of the required Service.
     * <p>
     * If require exact match is true, then the major version of the Service must match this valu
e. If requireExact
     * is false, then the major version of the Service must be at least this value.
     *
     * @return the major version
     */
    int majorVersion() default 0;
    /**
     * Specifies the minor version of the required Service.
     * <p>
     * If require exact match is true, then the minor version of the Service must match this valu
e. If requireExact
     * is false, then the minor version of the Service must be at least this value.
     *
     * @return the major version
     */
    int minorVersion() default 0;
    /**
     * Specifies whether or not the major and minor version of the Service must match exactly.
     *
     * @return true if an exact match is required, false otherwise.
     */
    boolean requireExact() default false;
}
```

A.248 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/features/Resolver.java

```
package mil.sstaf.core.features;

import mil.sstaf.core.entity.EntityHandle;
import mil.sstaf.core.util.Injector;
import mil.sstaf.core.util.RNGUtilities;
import mil.sstaf.core.configuration.SSTAFConfiguration;
import mil.sstaf.core.util.SSTAFException;
import org.apache.commons.math3.random.MersenneTwister;
import org.apache.commons.math3.random.RandomGenerator;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.InvocationTargetException;
import java.util.*;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.ConcurrentMap;

import static mil.sstaf.core.features.Loaders.getHelpWithServices;
import static mil.sstaf.core.features.Loaders.load;
import static mil.sstaf.core.util.ReflectionUtils.getAllFields;

/**
 * Recursively resolves {@code Feature} dependencies.
 */
public class Resolver {

    private static final Logger logger = LoggerFactory.getLogger(Resolver.class);
    private final Map<FeatureSpecification, Feature> featureCache;
    private final Map<String, ? extends FeatureConfiguration> configurations;
    private final RandomGenerator generator;
    private final EntityHandle owner;
    private final ModuleLayer moduleLayer;

    /**
     * Constructor
     *
     * @param featureCache   a cache of loaded Features
     * @param configurations the configuration map loaded from the Entity.
     * @param owner          an {@code EntityHandle} to the {@code Entity} that will use this {@c
ode Feature}
     * @param seed           top-level seed for stochastic configurations
     */
    public Resolver(final Map<FeatureSpecification, Feature> featureCache,
                    final Map<String, ? extends FeatureConfiguration> configurations,
                    final EntityHandle owner,
                    final long seed,
                    final ModuleLayer moduleLayer) {
        this.featureCache = featureCache;
        this.configurations = configurations;
        this.owner = owner;
        this.generator = new MersenneTwister(seed);
        this.moduleLayer = moduleLayer;
    }

    /**
     * Easily creates a {@code Resolver} for tests
     *
     * @param configurations The configurations for the loaded {@code Features}
     * @return a new Resolver
     */
```

```
    public static Resolver makeTransientResolver(Map<String, ? extends FeatureConfiguration> conf
igurations) {
        EntityHandle eh = EntityHandle.makeDummyHandle();
        ConcurrentMap<FeatureSpecification, Feature> m = new ConcurrentHashMap<>();
        return new Resolver(m, configurations, eh, System.currentTimeMillis(),
                SSTAFConfiguration.getInstance().getRootLayer());
    }

    /**
     * Easily creates a Resolver for tests
     *
     * @return a new Resolver with an empty configuration object.
     */
    public static Resolver makeTransientResolver() {
        return makeTransientResolver(Map.of());
    }


    /**
     * Loads a {@code Feature} and all of its dependencies from a specification
     * <p>
     * If the
     *
     * @param specification the specification for the {@code Feature} or {@code Handler} or {@cod
e Agent}
     * @return the {@code Feature}
     */
    @SuppressWarnings("rawtypes")
    public Feature loadAndResolveDependencies(FeatureSpecification specification) {
        logger.debug("{} - Loading and resolving for {}", owner.getPath(), specification);
        Feature feature = findMatchInCache(specification, featureCache);
        if (feature == null) {
            logger.trace("{} - Feature not in cache, loading {}", owner.getPath(), specification)
;
            Optional<Feature> optionalFeature = load(Feature.class, specification, owner, moduleL
ayer);
            if (optionalFeature.isPresent()) {
                feature = optionalFeature.get();
                featureCache.put(FeatureSpecification.from(feature), feature);
                logger.debug("{} - Feature loaded and added to cache. {}", owner.getPath(), featu
reCache.keySet());
                resolveDependencies(feature);
            } else {
                String errorMessage = getHelpWithServices(specification, Feature.class);
                logger.error(errorMessage);
                throw new SSTAFException("Failed to load " +
                        specification + " as " + Feature.class);
            }
        }
        return feature;
    }

    /**
     * Resolves dependencies for an already-instantiated Feature
     *
     * @param feature the Feature for which dependencies must be loaded.
     */
    public void resolveDependencies(Feature feature) {
        logger.trace("Resolving dependencies for {}",
                feature.getClass().getName());
        var rv = loadRequiredServices(feature, featureCache);
        featureCache.putAll(rv);
        if (logger.isTraceEnabled()) {
            logger.trace("Contents of featureCache:");
            for (Map.Entry<FeatureSpecification, Feature> entry : featureCache.entrySet()) {
                logger.trace("    {} = {}", entry.getKey().toString(), entry.getValue().toString(
));
            }
```

```
        }
        logger.trace("{} - Configuring top-level Feature", feature.getName());
        Optional<? extends FeatureConfiguration> optConfig = getConfiguration(feature.getName());
        Class<? extends FeatureConfiguration> configClass = feature.getConfigurationClass();

        if (optConfig.isPresent()) {
            FeatureConfiguration featureConfiguration = optConfig.get();
            configureFeature(feature, configClass, featureConfiguration);
        } else {
            logger.info("No configuration was provided for {}", feature.getName());
        }
    }

    /**
     * Forces the {@code Feature} and {@code FeatureConfiguration} into agreement.
     *
     * @param feature       the {@code Feature} that is to be configured
     * @param clazz         the {@code Class} of the {@code FeatureConfiguration} object
     * @param configuration the {@code FeatureConfiguraion}
     * @param <T>           The type of the configuration
     */
    private <T extends FeatureConfiguration> void configureFeature(Feature feature,
                                                                   Class<? extends FeatureConfigu
ration> clazz,
                                                                   FeatureConfiguration configura
tion) {
        if (clazz.isAssignableFrom(configuration.getClass())) {
            long subSeed = RNGUtilities.generateSubSeed(generator);
            configuration.setSeed(subSeed);
            feature.configure(configuration);
        } else {
            String msg = String.format("The configuration provided for %s was a %s rather than th
e required %s",
                    feature.getName(), configuration.getClass().getName(),
                    clazz.getName());
            throw new SSTAFException(msg);
        }
    }

    /**
     * Retrieves a configuration for a particular Feature from the configuration map
     *
     * @param featureName the name of the {@code Feature}, used to select the configuration
     * @return the configuration or an empty JSONObject if a matching configuration is not found.
     */
    private Optional<? extends FeatureConfiguration> getConfiguration(final String featureName) {
        logger.trace("Getting configuration for {}", featureName);
        FeatureConfiguration configuration = configurations.get(featureName);
        return Optional.ofNullable(configuration);
    }

    /**
     * Scans all Fields looking for @Requires associated with Feature references
     *
     * @param target the Feature to examine
     * @return a List of Fields that are properly annotated.
     */
    private List<Field> getFieldsWithRequires(final Feature target) {
        logger.debug("{} - Scanning fields for Requires annotations on Feature types", target.get
Name());
        List<Field> requiresFields = new ArrayList<>();
        for (Field field : getAllFields(target.getClass())) {
            logger.trace("{} - Processing field {}", target.getName(), field.getName());

            if (field.isAnnotationPresent(Requires.class)) {
                logger.trace("{} - Processing Requires on field {}", target.getName(), field.getN
ame());
                Class<?> rawClass = field.getType();
```

```
                if (Feature.class.isAssignableFrom(rawClass)) {
                    requiresFields.add(field);

                } else {
                    logger.debug("Field {} is marked @Requires but the type is not a Feature type
", field.getName());
                }
            } else {
                logger.trace("{} - Field {} is not annotated", target.getName(), field.getName())
;
            }
        }
        logger.debug("{} - Found {} @Requires annotations, {}", target.getName(),
                requiresFields.size(), requiresFields);
        return requiresFields;
    }

    /**
     * Checks whether a field has been set
     *
     * @param target the Feature in which to look
     * @param field  the Field to check
     * @return true if it has been set or the Field cannot be accessed.
     */
    private boolean isFieldSet(Feature target, Field field) {
        try {
            boolean accessible = field.canAccess(target);
            if (!accessible) {
                field.setAccessible(true);
            }
            Object contents = field.get(target);
            field.setAccessible(accessible);
            return contents != null;
        } catch (IllegalAccessException e) {
            e.printStackTrace();
            return true;
        }
    }

    private Feature findMatchInCache(FeatureSpecification desired,
                                     Map<FeatureSpecification, Feature> cache) {
        if (logger.isTraceEnabled()) {
            logger.trace("Looking for match to spec {} in cache {}", desired, cache.keySet());
        }
        Feature match = null;
        for (Map.Entry<FeatureSpecification, Feature> entry : cache.entrySet()) {
            FeatureSpecification have = entry.getKey();
            if (desired.isSatisfiedBy(have)) {
                match = entry.getValue();
                break;
            }
        }
        return match;
    }

    /**
     * For a Feature, load all of its required services, recurse to fill in the dependencies.
     *
     * @param target the {@code Feature} to inject the services into.
     */
    private Map<FeatureSpecification, Feature>
    loadRequiredServices(Feature target, Map<FeatureSpecification, Feature> parentCache) {
        logger.trace("Loading required services for {}:{}",
                owner.getPath(), target.getName());
        Map<FeatureSpecification, Feature> myCache = new HashMap<>(parentCache);

        for (Field field : getFieldsWithRequires(target)) {
```

```
            logger.trace("{}:{} - Processing Requires field {}",
                    owner.getPath(), target.getName(), field.getName());

            Requires requires = field.getAnnotation(Requires.class);
            Class<?> rawClass = field.getType();

            if (isFieldSet(target, field)) {
                logger.trace("{}:{} - Skipping field {} because it is already set",
                        owner.getPath(), target.getName(), field.getName());
            } else {
                var clazz = rawClass.asSubclass(Feature.class);
                FeatureSpecification spec = FeatureSpecification.from(requires, clazz);

                Feature toInject = findMatchInCache(spec, myCache);
                if (toInject != null) {
                    logger.debug("{}:{}/{} Found match in cache, reusing {}",
                            owner.getPath(), target.getName(), field.getName(), spec);
                } else {
                    logger.debug("{}:{}/{} Getting new instantiation of {}",
                            owner.getPath(), target.getName(), field.getName(), spec);
                    final Feature newlyLoaded = Loaders.loadAsRef(clazz, spec.featureName, spec.m
ajorVersion,
                            spec.minorVersion, spec.requireExact, moduleLayer);

                    if (newlyLoaded == null) {
                        String identifier = spec.featureName == null || spec.featureName.length()
== 0
                                ? spec.featureClass.getName() : spec.featureName;
                        throw new SSTAFException("In "
                                + owner.getPath()
                                + ", Resolver could not load an implementation for '"
                                + identifier + "' into feature '" + target.getName()
                                + "', field '" + field.getName() + "'");
                    }

                    //
                    // Since this is a new load, recurse to fill it in.
                    //
                    FeatureSpecification fs = FeatureSpecification.from(newlyLoaded);
                    myCache.put(fs, newlyLoaded);
                    logger.trace("{}:{} Registered {} in cache, cache now holds {} entries",
                            owner.getPath(), target.getName(), fs, myCache.size());

                    logger.trace("{}:{} - Recursing now for {}",
                            owner.getPath(), target.getName(), newlyLoaded.getName());
                    var childCache = loadRequiredServices(newlyLoaded, myCache);
                    myCache.putAll(childCache);
                    logger.trace("{}:{} - Back from processing {}, cache now holds {} entries",
                            owner.getPath(), target.getName(), newlyLoaded.getName(), myCache.siz
e());

                    //
                    // Inject the ownerHandle into the feature. This must be done before configur
ation
                    // or an NPE might occur. Some feature loggers reference owner paths.
                    //
                    logger.trace("{}:{} - Injecting owner handle {}",
                            owner.getPath(), target.getName(), owner.getPath());
                    Injector.inject(newlyLoaded, owner);
                    //
                    // TODO
                    //
                    Class<? extends FeatureConfiguration> fc = newlyLoaded.getConfigurationClass(
);

                    Optional<? extends FeatureConfiguration> optConfig = getConfiguration(newlyLo
aded.getName());
```

```
                    optConfig.ifPresentOrElse(configuration -> {
                                logger.trace("{}:{} - Configuring {} with {}",
                                        owner.getPath(), target.getName(), newlyLoaded.getName(),
                                        configuration);
                                configureFeature(newlyLoaded, fc, configuration);
                            },
                            () -> {
                                logger.debug("{}:{} - No configuration provided for {}, using def
ault",
                                        owner.getPath(), target.getName(), newlyLoaded.getName())
;

                                try {
                                    Constructor<? extends FeatureConfiguration> constructor = fc.
getConstructor();
                                    FeatureConfiguration config = constructor.newInstance();
                                    configureFeature(newlyLoaded, fc, config);

                                } catch (NoSuchMethodException |
                                        InvocationTargetException |
                                        InstantiationException |
                                        IllegalAccessException e) {
                                    e.printStackTrace();
                                }


                            }

                    );

                    toInject = newlyLoaded;
                }
                //
                // Inject the service into the field.
                //
                logger.debug("{} - Injecting {} into field {}",
                        target.getName(), toInject.getName(), field.getName());
                Injector.injectField(target, field, toInject);
            }
        }
        return myCache;
    }

    /**
     * Scans the cache of loaded {@code Feature}s and returns those Features that match the
     * provided {@code Class}.
     *
     * @param clazz the Feature Class to match
     * @return a list of found Features
     */
    public <T extends Feature> List<T> getServicesFromCache(Class<? extends T> clazz) {
        List<T> found = new ArrayList<>();
        for (Feature f : featureCache.values()) {
            if (clazz.isAssignableFrom(f.getClass())) {
                found.add(clazz.cast(f));
            }
        }
        return found;
    }
}
```

A.249 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/features/ResourceManager.java

```
package mil.sstaf.core.features;
import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.node.ObjectNode;
import mil.sstaf.core.util.SSTAFException;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.net.URL;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.StandardCopyOption;
import java.security.CodeSource;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.*;
import java.util.jar.JarFile;
import java.util.jar.Manifest;
/**
 * Manages access to resources that must be extracted to disk for use.
 */
public class ResourceManager {
    public static final String HASH_ALGORITHM = "__hashAlgorithm";
    private static final Logger logger = LoggerFactory.getLogger(ResourceManager.class);
    private final Map<String, File> resourceFiles = new HashMap<>();
    private Path directory;
    private int numWritten = 0;
    /**
     * Constructor
     *
     * @param resourceOwner the {@code Class}
     */
    private ResourceManager(final Class<? extends Feature> resourceOwner, Path where) {
        new Extractor(resourceOwner).extract(where);
    }
    /**
     * Factory
     *
     * @param resourceOwner the {@code Class} that owns the resources.
     * @return a ResourceManager
     */
    public static ResourceManager getManager(final Class<? extends Feature> resourceOwner) {
        return new ResourceManager(resourceOwner, null);
    }
    /**
     * Factory
     *
     * @param resourceOwner   the {@code Class} that owns the resources.
     * @param installationDir where resources should be installed
     * @return a ResourceManager
     */
    public static ResourceManager getManager(final Class<? extends Feature> resourceOwner, Path i
nstallationDir) {
        return new ResourceManager(resourceOwner, installationDir);
    }
    /**
     * Extracts an item from the the resources package and writes it to a {@link File}.
     *
     * <p>
```

```
     * Some {@code Feature} implementation rely on external applications to perform their functio
n. These
     * applications might require access to items that are packaged as resources. Since files in
the jar
     * are not accessible it is necessary to extract the resources and write them as normal files
.
     * </p>
     *
     * @param resourceOwner the {@code Class} that owns the resource
     * @param resourceName  the name of the resource
     * @param tempDir       the directory into which the resource should be copied
     * @return a {@code File} that points to the extracted resource
     * @throws IOException if any of the {@code File} operations fail
     */
    public static File extractResource(Class<?> resourceOwner, String resourceName, Path tempDir)
throws IOException {
        Objects.requireNonNull(resourceOwner, "resourceOwner");
        Objects.requireNonNull(resourceName, "resourceName");
        Objects.requireNonNull(tempDir, "tempDir");
        Module ownerModule = resourceOwner.getModule();
        InputStream is = ownerModule.getResourceAsStream(resourceName);
        if (is == null) {
            logger.error("Resource {} was not found in module {}", resourceName,
                    ownerModule.getName());
            throw new IOException("Resource " + resourceName + " was not found");
        }
        Path path;
        if (resourceName.contains("/")) {
            int chopAt = resourceName.lastIndexOf('/');
            String resourceDirPath = resourceName.substring(0, chopAt);
            String terminal = resourceName.substring(chopAt + 1);
            Path dir = Path.of(String.valueOf(tempDir), resourceDirPath);
            Files.createDirectories(dir);
            path = Path.of(String.valueOf(dir), terminal);
        } else {
            path = Path.of(tempDir.toString(), resourceName);
        }
        Files.copy(is, path, StandardCopyOption.REPLACE_EXISTING);
        return path.toFile();
    }
    /**
     * Returns the number of resources extracted and written to disk.
     *
     * <p>
     * The number will range from 0 to the number of resources depending on whether or not a reso
urce is
     * already on disk.
     * </p>
     *
     * @return the number of files written
     */
    public int getNumWritten() {
        return numWritten;
    }
    /**
     * Returns the top-level directory into which the resources are written.
     *
     * @return the directory
     */
    public Path getDirectory() {
        return directory;
    }
    /**
     * Returns the {@code Map} that contains the association of resource names to {@code File}s.
     *
     * @return the {@code Map}
     */
    public Map<String, File> getResourceFiles() {
```

```
        return resourceFiles;
    }
    /**
     * Utility class for extracting and verifying
     */
    private class Extractor {
        private final Class<? extends Feature> resourceOwner;
        private final Map<String, String> checksums = new TreeMap<>();
        private MessageDigest messageDigest;
        private boolean locationIsNew;
        private List<String> resourcesToExtract;
        private String algorithm;
        private Extractor(Class<? extends Feature> owner) {
            this.resourceOwner = owner;
        }
        void extract(Path where) {
            try {
                determineResourcesToExtract();
                retrieveChecksums();
                this.messageDigest = MessageDigest.getInstance(algorithm);
                verifyChecksumExistForResources();
                makeExtractionLocation(where);
                writeResources();
            } catch (NoSuchAlgorithmException noSuchAlgorithmException) {
                throw new SSTAFException("The " + algorithm + " algorithm is not available");
            } catch (IOException exception) {
                throw new SSTAFException(exception);
            }
        }

        /**
         * Reads the manifest to determine what to pull.
         */
        void determineResourcesToExtract() throws IOException {
            CodeSource codeSource = resourceOwner.getProtectionDomain().getCodeSource();
            if (codeSource != null) {
                URL sourceJar = codeSource.getLocation();
                JarFile jarFile = new JarFile(sourceJar.getFile());
                Manifest mf = jarFile.getManifest();
                String value = mf.getMainAttributes().getValue("SSTAF-Resources-To-Extract");
                if (value == null) {
                    resourcesToExtract = List.of();
                } else {
                    resourcesToExtract = List.of(value.split(" "));
                }
            } else {
                resourcesToExtract = List.of();
            }
        }
        /**
         * Computes the checksum for the specified file using the {@code MessageDigest} algorithm
         * that was specified in the checksum file.
         *
         * @param file the file for which the checksum is to be computed
         * @return the checksum as a String
         * @throws IOException if something doesn't work
         */
        public String computeChecksum(final Path file) throws IOException {
            byte[] buffer = new byte[1024];
            int read;
            messageDigest.reset();
            InputStream inputStream = new FileInputStream(file.toFile());
            while ((read = inputStream.read(buffer)) > 0) {
                messageDigest.update(buffer, 0, read);
            }
            byte[] checksum = messageDigest.digest();
            return bytesToString(checksum);
        }
```

```
        /**
         * Converts the byte array from the {@code MessageDigest} into a hex string.
         *
         * @param bytes the bytes
         * @return a {@code String} containing hexadecimal values.
         */
        private String bytesToString(byte[] bytes) {
            StringBuilder hexString = new StringBuilder();
            for (byte aByte : bytes) {
                if ((0xff & aByte) < 0x10) {
                    hexString.append("0").append(Integer.toHexString((0xFF & aByte)));
                } else {
                    hexString.append(Integer.toHexString(0xFF & aByte));
                }
            }
            return hexString.toString();
        }
        /**
         * Creates or confirms directory for resource extraction.
         *
         * @param where the {@code Path} for the directory.
         */
        private void makeExtractionLocation(final Path where) {
            try {
                if (where == null) {
                    directory = Files.createTempDirectory(resourceOwner.getSimpleName());
                    locationIsNew = true;
                } else {
                    File f = where.toFile();
                    if (!(f.exists() && f.canRead() && f.canWrite())) {
                        throw new SSTAFException("Can't access specified resource directory " + w
here);
                    }
                    directory = where;
                    locationIsNew = false;
                }
            } catch (IOException exception) {
                throw new SSTAFException("Can't create temp directory");
            }
        }
        /**
         * Compares the checksum for a file on disk with the matching one in the checksum file.
         *
         * @param name the name of the resource
         * @param p    the path to the file on disk
         * @return true if the checksums match, false otherwise
         * @throws IOException if something bad happens.
         */
        private boolean compareChecksums(String name, Path p) throws IOException {
            String hashFromDisk = computeChecksum(p);
            String hashFromJar = checksums.get(name);
            return Objects.equals(hashFromJar, hashFromDisk);
        }
        /**
         * Writes all of the resources specified in the manifest to the desired directory
         *
         * @throws IOException if something bad happens
         */
        private void writeResources() throws IOException {
            for (String resource : resourcesToExtract) {
                File resourceFile;
                if (needToExtract(resource)) {
                    resourceFile = extractResource(resourceOwner, resource, directory);
                    Path p = Path.of(resourceFile.toString());
                    if (compareChecksums(resource, p)) {
                        ++numWritten;
                        resourceFiles.put(resource, resourceFile);
                    } else {
```

```
                        throw new SSTAFException("The checksum for the newly-written file doesn't
match");
                    }
                } else {
                    resourceFile = Path.of(directory.toString(), resource).toFile();
                }
                resourceFiles.put(resource, resourceFile);
            }
        }
        /**
         * Determines whether or not a resource needs to be written to disk.
         *
         * @param name the name of the resource
         * @return true if the resource must be written, false otherwise.
         * @throws IOException if something doesn't work
         */
        private boolean needToExtract(String name) throws IOException {
            if (locationIsNew) {
                return true;
            } else {
                Path p = Path.of(directory.toString(), name);
                File f = p.toFile();
                if (f.exists() && f.canRead()) {
                    if (!compareChecksums(name, p)) {
                        logger.warn("The hashes for " + name + " do not match");
                        return true;
                    } else {
                        return false;
                    }
                } else {
                    return true;
                }
            }
        }
        /**
         * Retrieves the checksums (hashes) that were recorded in the jar.
         */
        private void retrieveChecksums() {
            try {
                InputStream is = resourceOwner.getModule().getResourceAsStream("/checksums.json")
;
                ObjectMapper om = new ObjectMapper();
                JsonNode node = om.readTree(is);
                ObjectNode co = (ObjectNode) node;
                algorithm = co.get(HASH_ALGORITHM).asText();
                Iterator<String> iter = co.fieldNames();
                while(iter.hasNext()) {
                    String key = iter.next();
                    if (!Objects.equals(key, HASH_ALGORITHM)) {
                        String val = co.get(key).asText();
                        checksums.put(key, val);
                    }
                }
            } catch (IOException e) {
                throw new SSTAFException("Could not read checksum values in " + resourceOwner.get
Module().getName(),
                        e);
            }
        }
        /**
         * Checks that a checksum exists for each file that is to be extracted.
         */
        private void verifyChecksumExistForResources() {
            List<String> missing = new ArrayList<>();
            for (String file : resourcesToExtract) {
                if (!checksums.containsKey(file)) {
                    missing.add(file);
                }
```

```
            }
            if (!missing.isEmpty()) {
                throw new SSTAFException("Checksum table did not contain entries for: " + missing
);
            }
        }
    }

}
```

A.250 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/features/StringContent.java

```
package mil.sstaf.core.features;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
@SuperBuilder
@Jacksonized
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
@EqualsAndHashCode(callSuper = true)
public class StringContent extends HandlerContent {
    @Getter
    private final String message;
    public static StringContent of(String msg) {
        return builder().message(msg).build();
    }
}
```

A.251 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/features/package-info.java

```
/**
 * Utilities for creating and interacting with external helper applications.
 *
 * Some {@code Feature} implementations will use external applications to perform their functions
. This
 * package provides routines for interacting with those applications.
 *
 * In SSTAF 1.0, two helper application constructs are available. The first {@link }
 */
package mil.sstaf.core.features;
```

A.252 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/json/ArrayNodeWrapper.java

```
package mil.sstaf.core.json;

import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.node.ArrayNode;

import java.util.Iterator;
import java.util.Map;

public class ArrayNodeWrapper extends BaseNodeWrapper<ArrayNode> {


    protected ArrayNodeWrapper(NodeWrapper parent, ArrayNode node, ObjectMapperFactory objectMapp
erFactory, Map<String, JsonNode> referenceCache) {
        super(parent, node, objectMapperFactory, referenceCache);
    }

    @Override
    public void resolve() {
        Iterator<JsonNode> iter = myNode.elements();
        while (iter.hasNext()) {
            JsonNode value = iter.next();
           processNode(value);
        }
    }

    @Override
    public void replaceNode(JsonNode original, JsonNode replacement) {
        int index = -1;
        int i = 0;
        Iterator<JsonNode> iter = myNode.elements();
        while (iter.hasNext()) {
            JsonNode value = iter.next();
            if (value.equals(original)) {
                index = i;
                break;
            }
            ++i;
        }

        if (index < 0) {
            throw new JsonResolutionException("Failed to find original node [" + original.toStrin
g()  +
                    "] in list [" + myNode.toString() + "]");
        } else {
            myNode.set(index, replacement);
        }
    }
}
```

A.253 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/json/BaseNodeWrapper.java

```
package mil.sstaf.core.json;

import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.node.ArrayNode;
import com.fasterxml.jackson.databind.node.ObjectNode;

import java.nio.file.Path;
import java.util.Map;
import java.util.Objects;

public abstract class BaseNodeWrapper<T extends JsonNode> implements NodeWrapper {

    protected final NodeWrapper parent;

    protected final ObjectMapperFactory objectMapperFactory;
    protected T myNode;
    protected boolean resolved = false;
    protected Map<String, JsonNode> cache;
    private Path directory;

    protected BaseNodeWrapper(NodeWrapper parent, T node, ObjectMapperFactory objectMapperFactory
, Map<String, JsonNode> referenceCache) {
        this.parent = parent;
        this.objectMapperFactory = objectMapperFactory;
        this.myNode = node;
        this.cache = referenceCache;
        this.directory = parent == null ? Path.of(System.getProperty("user.dir")) : parent.getDir
ectory();
    }

    @Override
    public T getMyNode() {
        return myNode;
    }

    @Override
    public boolean isResolved() {
        return resolved;
    }

    @Override
    public NodeWrapper getParent() {
        return parent;
    }

    @Override
    public void setDirectory(Path dir) {
        this.directory = dir;
    }

    @Override
    public Path getDirectory() {
        return directory;
    }

    @Override
    public int refCount(String refName) {
        Objects.requireNonNull(refName);
        return parent == null ? 0 : parent.refCount(refName);
    }

    public void processNode(JsonNode root) {
```

```
        NodeWrapper wrapper = null;
        if (root.isArray()) {
            wrapper = new ArrayNodeWrapper(this, (ArrayNode) root, objectMapperFactory, cache);
        } else if (root.isObject()) {
            wrapper = new ObjectNodeWrapper(this, (ObjectNode) root, objectMapperFactory, cache);
        } else {
            if (root.isTextual()) {
                String s = root.textValue();
                if (s.endsWith(".json")) {
                    wrapper = new ReferenceWrapper(s, this, root, objectMapperFactory, cache);
                }
            }
        }
        if (wrapper != null) {
            wrapper.resolve();
        }

    }
}
```

A.254 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/json/EntityHandleSerializer.java

```
package mil.sstaf.core.json;
import com.fasterxml.jackson.core.JsonGenerator;
import com.fasterxml.jackson.databind.JavaType;
import com.fasterxml.jackson.databind.SerializerProvider;
import com.fasterxml.jackson.databind.ser.std.StdSerializer;
import mil.sstaf.core.entity.EntityHandle;
import java.io.IOException;
/**
 * Jackson serializer for {@code EntityHandle}s
 */
public class EntityHandleSerializer extends StdSerializer<EntityHandle> {
    public EntityHandleSerializer() {
        super(EntityHandle.class);
    }
    protected EntityHandleSerializer(Class<EntityHandle> t) {
        super(t);
    }
    protected EntityHandleSerializer(JavaType type) {
        super(type);
    }
    protected EntityHandleSerializer(Class<?> t, boolean dummy) {
        super(t, dummy);
    }
    protected EntityHandleSerializer(StdSerializer<?> src) {
        super(src);
    }
    @Override
    public void serialize(EntityHandle value, JsonGenerator gen, SerializerProvider provider) thr
ows IOException {
        String path = value.getPath();
        gen.writeString(path);
    }
}
```

A.255 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/json/JsonConfig.java

```
package mil.sstaf.core.json;
import java.nio.file.Path;
import java.util.Objects;
import java.util.concurrent.atomic.AtomicReference;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;
public class JsonConfig {
    private static final AtomicReference<JsonConfig> reference = new AtomicReference<>();
    private final Lock lock = new ReentrantLock();
    private Path rootDir;
    public static JsonConfig getInstance() {
        synchronized (reference) {
            JsonConfig jsonConfig = reference.get();
            if (jsonConfig == null) {
                jsonConfig = new JsonConfig();
                reference.set(jsonConfig);
            }
            return jsonConfig;
        }
    }
    private JsonConfig() {
        String cwd = System.getProperty("user.dir");
        rootDir = Path.of(cwd);
    }
    public void setRootDir(Path rootDir) {
        Objects.requireNonNull(rootDir);
        try {
            lock.lock();
            this.rootDir = rootDir;
        } finally {
            lock.unlock();
        }
    }
    public Path getRootDir() {
        try {
            lock.lock();
            return rootDir;
        } finally {
            lock.unlock();
        }
    }
}
```

A.256 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/json/JsonLoader.java

```
package mil.sstaf.core.json;
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.node.ObjectNode;
import mil.sstaf.core.util.SSTAFException;
import java.io.IOException;
import java.io.InputStream;
import java.io.Reader;
import java.nio.file.Path;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;
public class JsonLoader {
    private final ObjectMapper objectMapper;
    public JsonLoader() {
        objectMapper = new ObjectMapper();
    }
    private Class<?> getRootClass(ObjectNode rootNode) {
        if (rootNode.has("class")) {
            String className = rootNode.get("class").asText();
            try {
                return Class.forName(className);
            } catch (ClassNotFoundException e) {
                throw new SSTAFException("Could not load specified class '" + className + "'", e)
;
            }
        } else {
            throw new SSTAFException("JSON does not specify a class name\n" + rootNode.toPrettySt
ring());
        }
    }
    private Object loadObject(JsonNode jsonNode, Path sourceFile) throws JsonProcessingException
{
        if (jsonNode.isObject()) {
            Class<?> rootClass = getRootClass((ObjectNode) jsonNode);
            return getObjectInner(jsonNode, rootClass, sourceFile);
        } else {
            throw new SSTAFException("JSON did not define an object");
        }
    }
    private <T> T loadObject(JsonNode jsonNode, Class<T> asClass, Path sourceFile) throws JsonPro
cessingException {
        if (jsonNode.isObject()) {
            return getObjectInner(jsonNode, asClass, sourceFile);
        } else {
            throw new SSTAFException("JSON did not define an object");
        }
    }
    private <T> T getObjectInner(JsonNode jsonNode, Class<T> asClass, Path sourceFile) {
        ObjectNode topNode = (ObjectNode) jsonNode;
        ReferenceResolver referenceResolver = new ReferenceResolver(topNode, sourceFile, objectMa
pper);
        referenceResolver.processTree();
        try {
            return objectMapper.treeToValue(jsonNode, asClass);
        } catch (Exception e) {
            e.printStackTrace();
            throw new SSTAFException("Could not read object '" + jsonNode.toPrettyString() + "'",
e);
        }
    }
    public Object load(Reader reader, Path sourceDir) {
```

```
        // 2022.04.26 : RAB : Note: The 'fakeFile.json' gets stripped later.
        Path pathToFile;
        if (sourceDir.isAbsolute()) {
            Path tmp = sourceDir.normalize();
            pathToFile = Path.of(tmp.toString(), "fakeFile.json");
        } else {
            pathToFile = Path.of(System.getProperty("user.dir"), sourceDir.toString(), "fakeFile.
json").normalize().toAbsolutePath();
        }
        try {
            JsonNode jsonNode = objectMapper.readTree(reader);
            return loadObject(jsonNode, pathToFile);
        } catch (IOException e) {
            throw new SSTAFException("Could not load JSON", e);
        }
    }
    public Object load(Path filePath) {
        Path pathToFile;
        if (filePath.isAbsolute()) {
            pathToFile = filePath.normalize();
        } else {
            pathToFile = Path.of(System.getProperty("user.dir"), filePath.toString()).normalize()
.toAbsolutePath();
        }
        try {
            JsonNode jsonNode = objectMapper.readTree(pathToFile.toFile());
            return loadObject(jsonNode, pathToFile);
        } catch (IOException e) {
            throw new SSTAFException("Could not load JSON", e);
        }
    }
    public Object load(String jsonString, Path sourceFile) {
        try {
            JsonNode jsonNode = objectMapper.readTree(jsonString);
            return loadObject(jsonNode, sourceFile);
        } catch (IOException e) {
            throw new SSTAFException("Could not load JSON from '" + jsonString + "'", e);
        }
    }
    public Object load(JsonNode node, Path sourceDir) {
        try {
            return loadObject(node, sourceDir);
        } catch (IOException e) {
            throw new SSTAFException("Could not load JSON", e);
        }
    }
    public <T> T load(Reader reader, Class<T> asClass, Path sourceDir) {
        // 2022.04.26 : RAB : Note: The 'fakeFile.json' gets stripped later.
        Path pathToFile;
        if (sourceDir.isAbsolute()) {
            Path tmp = sourceDir.normalize();
            pathToFile = Path.of(tmp.toString(), "fakeFile.json");
        } else {
            pathToFile = Path.of(System.getProperty("user.dir"), sourceDir.toString(), "fakeFile.
json").normalize().toAbsolutePath();
        }
        try {
            JsonNode jsonNode = objectMapper.readTree(reader);
            return loadObject(jsonNode, asClass, pathToFile);
        } catch (IOException e) {
            throw new SSTAFException("Could not load JSON", e);
        }
    }
    public <T> T load(Path filePath, Class<T> asClass) {
        Path pathToFile;
        if (filePath.isAbsolute()) {
            pathToFile = filePath.normalize();
        } else {
```

```
            pathToFile = Path.of(System.getProperty("user.dir"), filePath.toString()).normalize()
.toAbsolutePath();
        }
        try {
            JsonNode jsonNode = objectMapper.readTree(pathToFile.toFile());
            return loadObject(jsonNode, asClass, pathToFile);
        } catch (IOException e) {
            throw new SSTAFException("Could not load JSON from '" + pathToFile + "'", e);
        }
    }
    public <T> T load(String jsonString, Class<T> asClass, Path sourceFile) {
        try {
            JsonNode jsonNode = objectMapper.readTree(jsonString);
            return loadObject(jsonNode, asClass, sourceFile);
        } catch (IOException e) {
            throw new SSTAFException("Could not load JSON from '" + jsonString + "'", e);
        }
    }
    public <T> T load(JsonNode node, Class<T> asClass, Path sourceFile) {
        try {
            return loadObject(node, asClass, sourceFile);
        } catch (IOException e) {
            throw new SSTAFException("Could not load JSON", e);
        }
    }
}
```

A.257 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/json/JsonNodeReference.java

```
package mil.sstaf.core.json;

import com.fasterxml.jackson.databind.JsonNode;
import java.nio.file.Path;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.Objects;
public class JsonNodeReference {

    private final JsonNode rootNode;
    private final Path directory;
    private final JsonNodeReference parent;
    private final String reference;
    private final Map<String, JsonNodeReference> cache;
    public JsonNodeReference(JsonNode node, String ref, JsonNodeReference parent) {
        Objects.requireNonNull(parent);
        this.rootNode = node;
        this.directory = parent.directory;
        this.parent = parent;
        this.reference = ref;
        this.cache = parent.cache;
    }
    public JsonNodeReference(JsonNode node, Path directory, Map<String, JsonNodeReference> cache)
{
        this.rootNode = node;
        this.directory = directory;
        this.parent = null;
        this.reference = null;
        this.cache = cache;
    }
    public JsonNode getRootNode() {
        return rootNode;
    }
    public Path getDirectory() {
        return directory;
    }
    public JsonNodeReference getParent() {
        return parent;
    }
    public String getReference() {
        return reference;
    }
    List<JsonNodeReference> identifyReferences() {
        List<JsonNodeReference> nodeContexts = new ArrayList<>();
        var iter = rootNode.fields();
        while (iter.hasNext()) {
            var entry = iter.next();
            JsonNode node = entry.getValue();
            if (node.isTextual()) {
                String s = node.textValue();
                if (s.endsWith(".json")) {
                          nodeContexts.add(new JsonNodeReference(node, s, this));
                }
            }
        }
        return nodeContexts;
    }

}
```

A.258 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/json/JsonResolutionException.java

```
package mil.sstaf.core.json;
import mil.sstaf.core.util.SSTAFException;
public class JsonResolutionException extends SSTAFException {
    public JsonResolutionException() {
    }
    public JsonResolutionException(String message) {
        super(message);
    }
    public JsonResolutionException(String message, Throwable cause) {
        super(message, cause);
    }
    public JsonResolutionException(Throwable cause) {
        super(cause);
    }
    public JsonResolutionException(String message, Throwable cause, boolean enableSuppression, bo
olean writableStackTrace) {
        super(message, cause, enableSuppression, writableStackTrace);
    }
}
```

A.259 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/json/JsonSourcePath.java

```
package mil.sstaf.core.json;
import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;
@Target(ElementType.FIELD)
@Retention(RetentionPolicy.RUNTIME)
public @interface JsonSourcePath {
}
```

A.260 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/json/NodeWrapper.java

```
package mil.sstaf.core.json;
import com.fasterxml.jackson.databind.JsonNode;
import java.nio.file.Path;
/**
 * Interface for objects that wrap {@code JsonNode}s and help resolve {@code JsonNode}
 * graphs that contain references.
 */
public interface NodeWrapper {
    /**
     * Provides the {@code JsonNode} that is wrapped by this {@code NodeWrapper}
     * @return the {@code JSONNode}
     */
    JsonNode getMyNode();
    /**
     * Returns the whether the {@code NodeWrapper} and its child graph has been resolved.
     * @return True if the graph has been resolved, false otherwise.
     */
    boolean isResolved();
    /**
     * Provides the parent object
     * @return the parent.
     */
    NodeWrapper getParent();
    /**
     * Processes this {@code JsonNode}, creates {@code JSONWrapper}s for each one and resolves
     * any references.
     */
    void resolve();
    /**
     * Scans up the graph to determine the number of times the reference has been loaded. This
     * determines if the graph is cyclic. A count rather that a boolean is used to enable limited
     * cycles to be processed if desired, eventually.
     *
     * @param refName the name of the reference to check
     * @return the number of times the reference is found.
     */
    int refCount(String refName);
    void setDirectory(Path dir);
    /**
     * Returns the {@code Path} for the current working directory.
     * @return
     */
    Path getDirectory();
    /**
     * Replaces the original {@code JsonNode} with the replacement {@code JsonNode}
     * @param original the original {@code JsonNode}
     * @param replacement the {@code JsonNode} with which to replace it.
     */
    void replaceNode(JsonNode original, JsonNode replacement);
}
```

A.261 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/json/ObjectMapperFactory.java

```
package mil.sstaf.core.json;
import com.fasterxml.jackson.databind.ObjectMapper;
import java.nio.file.Path;
/**
 * Used to configure JsonContext and JsonReferenceProcessor
 */
public interface ObjectMapperFactory {
    /**
     * Creates configured {@link ObjectMapper}
     *
     * @param path the {@link Path}
     * @return configured {@link ObjectMapper}
     */
    public ObjectMapper create(Path path);
}
```

A.262 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/json/ObjectNodeWrapper.java

```
package mil.sstaf.core.json;

import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.node.ObjectNode;

import java.nio.file.Path;
import java.util.HashSet;
import java.util.Map;
import java.util.Set;

public class ObjectNodeWrapper extends BaseNodeWrapper<ObjectNode> {

    protected ObjectNodeWrapper(NodeWrapper parent, ObjectNode node, ObjectMapperFactory objectMa
pperFactory, Map<String, JsonNode> referenceCache) {
        super(parent, node, objectMapperFactory, referenceCache);
    }

    protected ObjectNodeWrapper(ObjectNode node, ObjectMapperFactory objectMapperFactory, Map<Str
ing, JsonNode> referenceCache, Path workingDir) {
        super(null, node, objectMapperFactory, referenceCache);
        if (workingDir.isAbsolute()) {
            this.setDirectory(workingDir.normalize());
        } else {
            String userDir = System.getProperty("user.dir");
            Path p = Path.of(userDir, workingDir.toString()).normalize().toAbsolutePath();
            setDirectory(p);
        }
    }

    @Override
    public void resolve() {
        var iter = myNode.fields();
        Set<String> keys = new HashSet<>();
        while (iter.hasNext()) {
            Map.Entry<String, JsonNode> entry = iter.next();
            JsonNode value = entry.getValue();
            processNode(value);
            keys.add(entry.getKey());
        }
        resolved = true;

        // 2022.04.14 : RAB : kludge to inject source directory info into
        //                    objects during deserialization. Currently
        if (addDirectoryInfo(keys)) {
            myNode.put("sourceDir", getDirectory().toString());
        }
        ;
    }

    /**
     * Determine if the "sourceDir" property should be injected
     *
     * @param keys the keys in the object, used to determine the type
     * @return true if "sourceDir" should be added.
     */
    private boolean addDirectoryInfo(Set<String> keys) {
        return keys.contains("modules") && keys.contains("modulePaths");
    }

    @Override
    public void replaceNode(JsonNode original, JsonNode replacement) {
        var iter = myNode.fields();
```

```
        String key = null;
        while (iter.hasNext()) {
            Map.Entry<String, JsonNode> entry = iter.next();
            JsonNode value = entry.getValue();
            if (value.equals(original)) {
                key = entry.getKey();
                break;
            }
        }
        if (key != null) {
            myNode.set(key, replacement);
        }
    }
}
```

A.263 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/json/ReferenceResolver.java

```
package mil.sstaf.core.json;
import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.node.ArrayNode;
import com.fasterxml.jackson.databind.node.ObjectNode;
import mil.sstaf.core.util.SSTAFException;
import java.io.IOException;
import java.nio.file.Path;
import java.util.*;
import java.util.concurrent.ConcurrentHashMap;
public class ReferenceResolver {
    private final ObjectNode rootNode;
    private final Path sourceFile;
    private final Path baseDir;
    private ObjectMapper objectMapper;
    public ReferenceResolver(ObjectNode rootNode, Path sourceFile, ObjectMapper objectMapper) {
        Objects.requireNonNull(rootNode, "rootNode");
        Objects.requireNonNull(sourceFile, "sourceFile");
        if (sourceFile.isAbsolute()) {
            this.sourceFile = sourceFile.normalize();
        } else {
            String userDir = System.getProperty("user.dir");
            Path p = Path.of(userDir, sourceFile.toString()).normalize().toAbsolutePath();
            this.sourceFile = p;
        }
        baseDir = this.sourceFile.getParent();
        this.rootNode = rootNode;
        this.objectMapper = objectMapper;
    }
    public void processTree() {
        Deque<Path> pathDeque = new ArrayDeque<>();
        Map<Path, JsonNode> cache = new ConcurrentHashMap<>();
        pathDeque.push(sourceFile);
        processNode(rootNode, baseDir, pathDeque, cache);
    }
    private void processNode(JsonNode node, Path cwd, Deque<Path> paths, Map<Path, JsonNode> cach
e) {
        if (node.isObject()) {
            ObjectNode objectNode = (ObjectNode) node;
            processObject(objectNode, cwd, paths, cache);
        } else if (node.isArray()) {
            ArrayNode arrayNode = (ArrayNode) node;
            processArray(arrayNode, cwd, paths, cache);
        }
    }
    private boolean addDirectoryInfo(Set<String> keys) {
        return keys.contains("modules") && keys.contains("modulePaths");
    }
    private boolean isReference(JsonNode node) {
        if (node.isTextual()) {
          return node.asText().endsWith(".json");
        }
        return false;
    }
    private Path referenceToPath(JsonNode refNode, Path cwd) {
        Objects.requireNonNull(refNode, "refNode");
        Objects.requireNonNull(cwd, "cwd");
        if (refNode.isTextual()) {
            String ref = refNode.asText();
            Path raw = Path.of(cwd.toString(), ref);
            Path normalized = raw.normalize();
            Path absolute = normalized.toAbsolutePath();
```

```
            return absolute;
        } else {
            throw new SSTAFException("node is not text");
        }
    }
    private void processObject(ObjectNode objectNode, Path cwd, Deque<Path> paths, Map<Path, Json
Node> cache) {
        Set<String> keys = new HashSet<>();
        var iter = objectNode.fields();
        Map<String, JsonNode> replacements = new HashMap<>();
        while (iter.hasNext()) {
            Map.Entry<String, JsonNode> entry = iter.next();
            JsonNode value = entry.getValue();
            if (isReference(value)) {
                Path refPath = referenceToPath(value, cwd);
                checkCircularity(refPath, paths);
                JsonNode replacement = loadReference(refPath);
                cache.put(refPath, replacement);
                replacements.put(entry.getKey(), replacement);
                paths.push(refPath);
                Path refWorkingDir = refPath.getParent();
                processNode(replacement, refWorkingDir, paths, cache);
                paths.pop();
            } else {
                processNode(value, cwd, paths, cache);
            }
            keys.add(entry.getKey());
        }
        // After the original contents have been processed, replace
        // any references with their loaded nodes.
        objectNode.setAll(replacements);
        // 2022.04.14 : RAB : kludge to inject source directory info into
        //                    objects during deserialization. Currently
        if (addDirectoryInfo(keys)) {
            objectNode.put("sourceDir", cwd.toString());
        }
    }
    private void processArray(ArrayNode arrayNode, Path cwd, Deque<Path> paths, Map<Path, JsonNod
e> cache) {
        Map<Integer, JsonNode> replacements = new HashMap<>();
        int i = 0;
        Iterator<JsonNode> iter = arrayNode.elements();
        while (iter.hasNext()) {
            JsonNode value = iter.next();
            if (isReference(value)) {
                Path refPath = referenceToPath(value, cwd);
                checkCircularity(refPath, paths);
                paths.push(refPath);
                JsonNode replacement = loadReference(refPath);
                cache.put(refPath, replacement);
                replacements.put(i, replacement);
                Path refWorkingDir = refPath.getParent();
                processNode(replacement, refWorkingDir, paths, cache);
                paths.pop();
            } else {
                processNode(value, cwd, paths, cache);
            }
            ++i;
        }
        // After the original contents have been processed, replace
        // any references with their loaded nodes.
        for (var entry : replacements.entrySet()) {
            arrayNode.set(entry.getKey(), entry.getValue());
        }
    }
    private void checkCircularity(Path refPath, Deque<Path> stack) {
        if (stack.contains(refPath)) {
            throw new JsonResolutionException("Circular reference to '" + refPath + "'");
```

```
        }
    }
    private JsonNode loadReference(Path reference) {
        try {
            return objectMapper.readTree(reference.toFile());
        } catch (IOException e) {
            throw new JsonResolutionException("Could not load reference '" + reference +"'", e);
        }
    }
}
```

A.264 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/json/ReferenceWrapper.java

```
package mil.sstaf.core.json;

import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;

import java.io.IOException;
import java.nio.file.Path;
import java.util.Map;
import java.util.Objects;

public class ReferenceWrapper extends BaseNodeWrapper<JsonNode> {

    private final String ref;
    private Path pathToRef;

    public ReferenceWrapper(String ref, NodeWrapper parent, JsonNode node,
                            ObjectMapperFactory objectMapperFactory,
                            Map<String, JsonNode> referenceCache) {
        super(parent, node, objectMapperFactory, referenceCache);
        this.ref = ref;
        this.pathToRef = makeRegPath(ref);
        this.setDirectory(pathToRef.getParent());
    }

    public Path makeRegPath(String ref) {
        if (ref.startsWith("/")) {
            return Path.of(ref).normalize().toAbsolutePath();
        } else {
            return Path.of(getDirectory().toString(), ref).normalize().toAbsolutePath();
        }
    }

    @Override
    public void resolve() {

        if (parent.refCount(ref) == 0) {
            String pathString = pathToRef.toString();
            JsonNode replacementNode;
            if (cache.containsKey(pathString)) {
                replacementNode = cache.get(pathString);
            } else {
                replacementNode = loadAndResolve(pathString);
                cache.put(pathString, replacementNode);
            }
            parent.replaceNode(myNode, replacementNode);
            resolved = true;
        } else {
            throw new JsonResolutionException("Circular reference '" + ref + "'");
        }
    }

    private JsonNode loadAndResolve(String path) {
        try {
            Path p = Path.of(path);
            ObjectMapper objectMapper = objectMapperFactory.create(p);
            JsonNode replacementNode = objectMapper.readTree(p.toFile());
            processNode(replacementNode);
            return replacementNode;
        } catch (IOException ioException) {
            throw new JsonResolutionException("Could not read reference '"
                    + ref
                    + "', which was resolved to '"
```

```
                    + path + "'", ioException);
        }
    }

    @Override
    public int refCount(String refName) {
        Objects.requireNonNull(refName);
        return super.refCount(refName) + (refName.equals(ref) ? 1 : 0);
    }

    @Override
    public void replaceNode(JsonNode original, JsonNode replacement) {

    }

}
```

A.265 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/module/ModuleLayerDefinition.java

```
package mil.sstaf.core.module;
import lombok.Builder;
import lombok.Getter;
import lombok.extern.jackson.Jacksonized;
import java.nio.file.Path;
import java.util.Set;
/**
 * Defines the configuration of a {@link ModuleLayer}.
 *
 * Multiple module layers can be defined. A root layer can be defined for
 * the entire SSTAF infrastructure by adding a {@code ModuleLayerDefinition
 * to the {@code SSTAFConfiguration}. Additionally, each {@code Entity} can
 * define a {@code ModuleLayer} that will be used to load it's {@code Featues}.
 *
 * 2022.04.26 : RAB : TODO: Need more test coverage for Modules and Class
 *                          loading scenarios.
 */
@Builder
@Jacksonized
public final class ModuleLayerDefinition {
    /**
     * The directory from which this {@code ModuleLayerDefinition} was loaded.
     * Used to resolve relative module paths.
     *
     * 2022.04.26 : RAB :
     * Note: The value for sourceDir is injected by {@code JSONLoader}. It is
     *       hard-wired to recognize {@code ModuleLayerDefinition} objects by
     *       their fields. This is probably the only truly awful hack in
     *       the system.
     */
    @Getter
    private final Path sourceDir;
    /**
     * The modules to include in the new layer.
     */
    @Getter
    private final Set<String> modules;
    /**
     * The paths to search for the modules.
     */
    @Getter
    private final Set<Path> modulePaths;
}
```

A.266 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/module/ModuleLayerSupport.java

```
package mil.sstaf.core.module;
import mil.sstaf.core.util.SSTAFException;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import java.lang.module.Configuration;
import java.lang.module.FindException;
import java.lang.module.ModuleFinder;
import java.nio.file.Path;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
public class ModuleLayerSupport {
    private static final Logger logger = LoggerFactory.getLogger(ModuleLayerSupport.class);

    public static ModuleLayer makeModuleLayer(ModuleLayer parentLayer,
                                              ModuleLayerDefinition def,
                                              ClassLoader parentLoader) {

        ModuleLayer moduleLayer;
        if (def == null) {
            logger.debug("Reusing module layer from SSTAFConfiguration");
            moduleLayer = parentLayer;
        } else if (def.getModules() == null || def.getModules().isEmpty()) {
            logger.debug("Reusing module layer from SSTAFConfiguration");
            moduleLayer = parentLayer;
        } else {
            List<Path> paths;
            if (def.getModulePaths() == null) {
                paths = List.of();
            } else {
                paths = new ArrayList<>(def.getModulePaths().size());
                for (Path p : def.getModulePaths()) {
                    if (p.isAbsolute()) {
                        paths.add(p);
                    } else {
                        Path baseDir = def.getSourceDir() == null ?
                                Path.of(System.getProperty("user.dir")) :
                                def.getSourceDir();
                        Path expanded = Path.of(baseDir.toString(), p.toString()).normalize();
                        paths.add(expanded);
                    }
                }
            }
            for (Path p : paths) {
                if (!p.toFile().exists()) {
                    throw new SSTAFException("Module path '" + p + "' does not exist.");
                }
            }
            try {
                logger.debug("Creating new module layer. Parent ClassLoader is {}", parentLoader)
;
                Path[] pathArray = new Path[paths.size()];
                paths.toArray(pathArray);
                if (logger.isDebugEnabled()) {
                    logger.debug("paths are {}", Arrays.toString(pathArray));
                    logger.debug("modules are {}", def.getModules());
                }
                ModuleFinder moduleFinder = ModuleFinder.of(pathArray);
                Configuration configuration = Configuration.resolveAndBind(moduleFinder,
                        List.of(parentLayer.configuration()),
                        ModuleFinder.of(), def.getModules());
```

```
                moduleLayer = parentLayer.defineModulesWithOneLoader(configuration, parentLoader)
;
                logger.debug("moduleLayer is {}", moduleLayer);
            } catch (FindException findException) {
                throw new SSTAFException("Could not find a module given paths " + paths, findExce
ption);
            }
        }
        return moduleLayer;
    }
}
```

A.267 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/state/LabeledState.java

```
package mil.sstaf.core.state;
import lombok.Data;
import lombok.experimental.SuperBuilder;
import mil.sstaf.core.entity.EntityHandle;
@Data
@SuperBuilder
public abstract class LabeledState implements State {
    public final EntityHandle owner;
    public final long timestamp_ms;
}
```

A.268 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/state/State.java

```
package mil.sstaf.core.state;
/**
 * Marker interface for State records
 */
public interface State {
}
```

A.269 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/state/StateProperty.java

```
package mil.sstaf.core.state;
import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;
/**
 * Annotation for specifying a property in a {@code LabeledState} object that can be recorded
 * <p>
 * The annotation can specify the name, major version, minor version and exact match
 * requirement for the Service to be injected.
 */
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface StateProperty {
    /**
     * Specifies the label to use for reporting the value
     *
     * @return the label if specified or an empty {@code String} if it wasn't
     */
    String headerLabel() default "";
}
```

A.270 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/util/CompletionServiceAdapter.java

```
package mil.sstaf.core.util;
import java.util.List;
import java.util.concurrent.*;
import java.util.function.Consumer;
/**
 * Simple utility to processing tasks concurrently and processing the results as they become
 * available
 */
public class CompletionServiceAdapter {
    /**
     * Simplifies the idiomatic pattern for using a CompletionService into a single method.
     *
     * @param callables       a List of the Callables to process
     * @param executorService the Executor that will process the Callable tasks.
     * @param consumer        a Consumer that processes each Future
     * @param <T>             the return type of the call() method in the Callable.
     */
    public static <T> void processCompletions(final List<Callable<T>> callables,
                                              final ExecutorService executorService,
                                              Consumer<T> consumer) {
        try {
            CompletionService<T> completionService = new ExecutorCompletionService<T>(executorSer
vice);
            callables.forEach(completionService::submit);
            Future<T> future;
            int remaining = callables.size();
            do {
                future = completionService.take();
                T value = future.get();
                --remaining;
                consumer.accept(value);
            } while (remaining > 0);
        } catch (InterruptedException | ExecutionException e) {
            throw new SSTAFException("Processing failed!", e);
        }
    }
}
```

A.271 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/util/FactoryProvider.java

```
package mil.sstaf.core.util;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface FactoryProvider {

}
```

A.272 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/util/Formatter.java

```
package mil.sstaf.core.util;
/**
 * Utility methods for pretty-printing things.
 */
public class Formatter {
    public static String millisToDHMS(long time_ms) {
        long toSeconds = 1000;
        long toMinutes = 60 * toSeconds;
        long toHours = 60 * toMinutes;
        long toDays = 24 * toHours;
        long days = time_ms / toDays;
        long rem1 = time_ms % toDays;
        long hours = rem1 / toHours;
        long rem2 = rem1 % toHours;
        long minutes = rem2 / toMinutes;
        long rem3 = rem2 % toMinutes;
        double seconds = (double) rem3 / (double) toSeconds;
        return String.format("%d:%d:%d:%.3f", days, hours, minutes, seconds);
    }
}
```

A.273 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/util/Injected.java

```
package mil.sstaf.core.util;
import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;
/**
 * Annotation for marking dependency injection points.
 */
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.FIELD)
public @interface Injected {
    /**
     * Provides for a name or label to be associated with an injection point. This allows for mul
tiple
     * injection points of the same type to be differentiated.
     *
     * @return the name of the field.
     */
    String name() default "";
}
```

A.274 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/util/Injector.java

```
package mil.sstaf.core.util;
import mil.sstaf.core.features.Requires;
import org.slf4j.LoggerFactory;
import org.slf4j.Logger;
import java.lang.reflect.Field;
import java.util.Arrays;
import java.util.Objects;
import static mil.sstaf.core.util.ReflectionUtils.getAllFields;
/**
 * Utility class for performing dependency injection. Given the very limited use-cases for DI in
SSTAF,
 * this was much simpler than using Guice or Spring.
 */
public class Injector {
    private static final Logger logger = LoggerFactory.getLogger(Injector.class);
    /**
     * Injects an object into a specific field.
     *
     * @param target         the {@code Object} containing the Field.
     * @param field          the {@code Field} into which the implementation object is to be inje
cted.
     * @param implementation the object ot be injected.
     * @param <T>            the type of the implementation
     */
    public static <T> void injectField(final Object target, final Field field, final T implementa
tion) {
        try {
            final boolean accessible = field.canAccess(target);
            if (!accessible) {
                field.setAccessible(true);
            }
            field.set(target, implementation);
            field.setAccessible(accessible);
        } catch (IllegalAccessException e) {
            //
            // This block should not be reachable since using setAccessible(true)
            // makes private values writeable.,
            //
            e.printStackTrace();
        }
    }
    /**
     * Injects an implementation into the specified target Object.
     *
     * @param target         the {@code Object} containing the Field.
     * @param implementation the object ot be injected.
     * @param <T>            the type of the implementation
     */
    public static <T> void inject(final Object target, final T implementation) {
        Objects.requireNonNull(target, "Target is null");
        Objects.requireNonNull(implementation, "Implementation is null");
        Class<?> c = target.getClass();
        getAllFields(c).forEach(field -> {
            if (field.getAnnotation(Injected.class) != null ||
                    field.getAnnotation(Requires.class) != null) {
                Class<?> type = field.getType();
                if (type.isAssignableFrom(implementation.getClass())) {
                    injectField(target, field, implementation);
                }
            }
        });
    }
```

```
    /**
     * Injects an implementation into the specified target Object at the field marked with specif
ied field name.
     *
     * @param target         the {@code Object} containing the Field.
     * @param fieldName      the name/label of the injection point.
     * @param implementation the object ot be injected.
     * @param <T>            the type of the implementation.
     */
    public static <T> void inject(final Object target, final String fieldName, final T implementa
tion) {
        Objects.requireNonNull(target, "Target is null");
        Objects.requireNonNull(fieldName, "Field name is null");
        Objects.requireNonNull(implementation, "Implementation is null");
        Class<?> c = target.getClass();
        getAllFields(c).forEach(field -> {
            Injected injected = field.getAnnotation(Injected.class);
            if (injected != null && injected.name().equals(fieldName)) {
                Class<?> type = field.getType();
                if (type.isAssignableFrom(implementation.getClass())) {
                    logger.trace("Injecting {} into {}", implementation, field.getName());
                    injectField(target, field, implementation);
                }
            }
        });
    }

    /**
     * Injects multiple objects into a single target object
     *
     * @param target          the {@code Object} containing the Field.
     * @param implementations the objects ot be injected.
     */
    public static void injectAll(final Object target, final Object... implementations) {
        Arrays.asList(implementations).forEach(implementation -> {
            logger.trace("Injecting {} into {}", implementation, target);
            if (implementation != null) inject(target, implementation);
        });
    }
}
```

A.275 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/util/PairValueMap.java

```
package mil.sstaf.core.util;
import java.util.HashMap;
public class PairValueMap<K, V> extends HashMap<PairValueMap.KeyPair<K>, V> {
    static class KeyPair<K> {
        K key1;
        K key2;
        KeyPair(K key1, K key2) {
            this.key1 = key1;
            this.key2 = key2;
        }
        @Override
        public boolean equals(Object o) {
            if (this == o) return true;
            if (o == null || getClass() != o.getClass()) return false;
            KeyPair<?> keyPair = (KeyPair<?>) o;
            return (key1.equals(keyPair.key1) && key2.equals(keyPair.key2)
                    || key1.equals(keyPair.key2) && key2.equals(keyPair.key1));
        }
        @Override
        public int hashCode() {
            //
            // I want order independence
            //
            return key1.hashCode() * key2.hashCode();
        }
    }
}
```

A.276 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/util/PhysicalConstants.java

```
package mil.sstaf.core.util;
public class PhysicalConstants {
    public static final double GRAVITY = 9.80665; // m/s2
}
```

A.277 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/util/RNGUtilities.java

```
package mil.sstaf.core.util;
import org.apache.commons.math3.random.RandomGenerator;
/**
 * Utilities for random numbers.
 */
public class RNGUtilities {
    /**
     * Generates a new random number seed from a parent generator.
     * <p>
     * The purpose of this method is to repeatedly create seeds for randomized sub-problems
     * that are not immediately adjacent to the parent generator. For most generators, seeding wi
th
     * the next value from the parent nearly synchronizes the child with the parent, resulting in
     * overlaps in the draws. This method XORs two draws to try to get to a different part of the
     * random draw space.
     *
     * @param randomGenerator the parent random number generator
     * @return the seed for the sub-problem.
     */
    public static long generateSubSeed(final RandomGenerator randomGenerator) {
        long val1 = randomGenerator.nextLong();
        long val2 = randomGenerator.nextLong();
        return val1 ^ val2;
    }
}
```

A.278 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/util/ReflectionUtils.java

```
package mil.sstaf.core.util;
import java.lang.annotation.Annotation;
import java.lang.reflect.Field;
import java.lang.reflect.Method;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
/**
 * General-purpose utility methods for handling some reflective tasks.
 */
public class ReflectionUtils {
    private static void addFieldsForClass(Class<?> c, List<Field> l) {
        l.addAll(Arrays.asList(c.getDeclaredFields()));
    }
    public static List<Field> getAllFields(Class<?> c) {
        List<Field> fields = new ArrayList<>();
        addFieldsForClass(c, fields);
        Class<?> sc = c;
        while ((sc = sc.getSuperclass()) != null) {
            addFieldsForClass(sc, fields);
        }
        return fields;
    }
    public static <A extends Annotation> List<Method> getMethodsWithAnnotation(Class<?> clazz, Cl
ass<A> annotationClass) {
        List<Method> methods = new ArrayList<>(2);
        for (Method m : clazz.getDeclaredMethods()) {
            final A annotation = m.getAnnotation(annotationClass);
            if (annotation != null) {
                methods.add(m);
            }
        }
        return methods;
    }
    public static <A extends Annotation> List<Method> getMethodsWithAnnotationOrThrow(Class<?> cl
azz, Class<A> annotationClass) {
        List<Method> methods = getMethodsWithAnnotation(clazz, annotationClass);
        if (methods.size() > 0) {
            return methods;
        } else {
            String msg = String.format("Did not find a method annotated with '%s' in class '%s'",
                    annotationClass.getName(), clazz.getName());
            throw new SSTAFException(msg);
        }
    }
}
```

A.279 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/util/ResourceUtilities.java

```
package mil.sstaf.core.util;
import mil.sstaf.core.features.ResourceManager;
import java.io.File;
import java.io.IOException;
import java.io.InputStream;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.StandardCopyOption;
import java.util.Objects;
/**
 * Utilities for interacting with resources.
 * <p>
 *
 * @author Ron Bowers
 * @since 1.0
 */
public class ResourceUtilities {
    public static File extractResource(Class<?> resourceOwner, String resourceName, Path tempDir)
throws IOException {
    return ResourceManager.extractResource(resourceOwner, resourceName, tempDir);
    }
}
```

A.280 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/util/SSTAFException.java

```
package mil.sstaf.core.util;
public class SSTAFException extends RuntimeException {
    public SSTAFException(String message) {
        super(message);
    }
    public SSTAFException() {
    }
    public SSTAFException(String s, Throwable throwable) {
        super(s, throwable);
    }
    public SSTAFException(Throwable cause) {
        super(cause);
    }
    protected SSTAFException(String message, Throwable cause, boolean enableSuppression, boolean
writableStackTrace) {
        super(message, cause, enableSuppression, writableStackTrace);
    }
}
```

A.281 SSTAF/src/framework/mil.sstaf.core/src/main/java/mil/sstaf/core/util/Validation.java

```
package mil.sstaf.core.util;
import java.util.function.Predicate;
/**
 * Simple utility methods for validating values.
 */
public class Validation {
    /**
     * Tests the provided target value using the provided {@code Predicate}
     *
     * @param target    the value to test
     * @param predicate the {@code Predicate} to use to test the value
     * @param message   the message to include in the {@code} print if the test fails.
     * @param <T>       the type of the value to test.
     * @return the target value
     */
    public static <T> T require(T target, Predicate<T> predicate, String message) {
        if (!predicate.test(target)) throw new IllegalArgumentException(message);
        return target;
    }
    /**
     * Tests the provided target value using the provided {@code Predicate}
     *
     * @param target    the value to test
     * @param predicate the {@code Predicate} to use to test the value
     * @param <T>       the type of the value to test.
     * @return the target value
     */
    public static <T> T require(T target, Predicate<T> predicate) {
        return require(target, predicate, target + " is not valid");
    }
}
```

A.282 SSTAF/src/framework/mil.sstaf.core/src/main/java/module-info.java

```
import mil.sstaf.core.features.Agent;
import mil.sstaf.core.features.Feature;
import mil.sstaf.core.features.Handler;
/**
 * @author Ron Bowers
 * @uses Feature
 * @uses Handler
 * @uses Agent
 * <p>
 * Base module for the Soldier and Squad Trade Space Analysis Framework (SSTAF).
 * @since 1.0
 */
module mil.sstaf.core {
    exports mil.sstaf.core.entity;
    exports mil.sstaf.core.util;
    exports mil.sstaf.core.state;
    exports mil.sstaf.core.features;
    exports mil.sstaf.core.configuration;
    exports mil.sstaf.core.module;
    requires transitive commons.math3; // For RandomGenerator
    requires transitive com.fasterxml.jackson.databind;
    requires transitive lombok;
    requires org.slf4j;
    uses Feature;
    uses Handler;
    uses Agent;
    exports mil.sstaf.core.json;
    opens mil.sstaf.core.configuration to com.fasterxml.jackson.databind;
    opens mil.sstaf.core.module to com.fasterxml.jackson.databind;
    opens mil.sstaf.core.entity to com.fasterxml.jackson.databind;
    opens mil.sstaf.core.features to com.fasterxml.jackson.databind;
    opens mil.sstaf.core.json to com.fasterxml.jackson.databind;
}
```

A.283 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/configuration/SSTAFConfigurationTest.java

```
package mil.sstaf.core.configuration;

import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.node.ArrayNode;
import com.fasterxml.jackson.databind.node.ObjectNode;
import org.junit.jupiter.api.*;

import java.io.File;
import java.nio.file.Path;
import java.util.ArrayList;
import java.util.List;
import java.util.ServiceLoader;
import java.util.function.Predicate;

public class SSTAFConfigurationTest {


    public static final String PATH_STRING = "src/test/resources/moduleLayers";
    public static final Path SOURCE_FILE = Path.of(PATH_STRING, "config.json");

    @BeforeEach
    void b4each() {
        System.setProperty(SSTAFConfiguration.SSTAF_CONFIGURATION_PROPERTY, "");
        SSTAFConfiguration.reset();
    }

    @Nested
    @DisplayName("Confirm that the no-arg scenarios work")
    public class NoArgTests {

        @DisplayName("Confirm that invoking SSTAFContext.getRootLayer without init'ing returns th
e boot layer")
        @Test
        public void test0() {
            ModuleLayer ml = SSTAFConfiguration.getInstance().getRootLayer();
            Assertions.assertEquals(ModuleLayer.boot(), ml);
        }

        @DisplayName("Check that system property -> SSTAFConfigurationTest1.json works")
        @Test
        public void test1() {
            System.setProperty(SSTAFConfiguration.SSTAF_CONFIGURATION_PROPERTY,
                    "src/test/resources/moduleLayers/SSTAFConfigurationTest1.json");
            ModuleLayer ml = SSTAFConfiguration.getInstance().getRootLayer();
            Assertions.assertNotNull(ml);

            var sl = ServiceLoader.load(ml, Predicate.class);
            long matches = sl.stream().count();
            Assertions.assertEquals(2, matches);
        }

        @DisplayName("Check that system property -> SSTAFConfigurationTest2.json works")
        @Test
        public void test2() {
            System.setProperty(SSTAFConfiguration.SSTAF_CONFIGURATION_PROPERTY,
                    "src/test/resources/moduleLayers/SSTAFConfigurationTest2.json");
            ModuleLayer ml = SSTAFConfiguration.getInstance().getRootLayer();
            Assertions.assertNotNull(ml);

            var sl = ServiceLoader.load(ml, Predicate.class);
            long matches = sl.stream().count();
```

```
            Assertions.assertEquals(3, matches);
        }

    }

    @Nested
    @DisplayName("Confirm that SSTAFContext can be initialized from a configuration file")
    public class JSONFileTests {

        @DisplayName("Use SSTAFConfigurationTest1.json")
        @Test
        public void test1() {
            System.setProperty(SSTAFConfiguration.SSTAF_CONFIGURATION_PROPERTY,
                    "src/test/resources/moduleLayers/SSTAFConfigurationTest1.json");
            ModuleLayer ml = SSTAFConfiguration.getInstance().getRootLayer();
            Assertions.assertNotNull(ml);

            var sl = ServiceLoader.load(ml, Predicate.class);
            long matches = sl.stream().count();
            Assertions.assertEquals(2, matches);
        }

        @DisplayName("Use SSTAFConfigurationTest2.json")
        @Test
        public void test2() {
            System.setProperty(SSTAFConfiguration.SSTAF_CONFIGURATION_PROPERTY,
                    "src/test/resources/moduleLayers/SSTAFConfigurationTest2.json");
            ModuleLayer ml = SSTAFConfiguration.getInstance().getRootLayer();
            Assertions.assertNotNull(ml);

            var sl = ServiceLoader.load(ml, Predicate.class);
            long matches = sl.stream().count();
            Assertions.assertEquals(3, matches);
        }

        @DisplayName("Use SSTAFConfigurationTest3.json")
        @Test
        public void test3() {
            System.setProperty(SSTAFConfiguration.SSTAF_CONFIGURATION_PROPERTY,
                    "src/test/resources/moduleLayers/SSTAFConfigurationTest3.json");
            ModuleLayer ml = SSTAFConfiguration.getInstance().getRootLayer();
            Assertions.assertNotNull(ml);

            var sl = ServiceLoader.load(ml, Predicate.class);
            long matches = sl.stream().count();
            Assertions.assertEquals(4, matches);
        }

        @DisplayName("Confirm that subclassing SSTAFConfiguration and loading it works automagica
lly")
        @Test
        public void test4() {
            System.setProperty(SSTAFConfiguration.SSTAF_CONFIGURATION_PROPERTY,
                    "src/test/resources/moduleLayers/SSTAFConfigurationTest4.json");
            ModuleLayer ml = SSTAFConfiguration.getInstance().getRootLayer();
            Assertions.assertNotNull(ml);
            SSTAFConfigurationToo sstafConfigurationToo = (SSTAFConfigurationToo)
                    SSTAFConfiguration.getInstance();

            Assertions.assertEquals("This is something more!", sstafConfigurationToo.getMore());

            var sl = ServiceLoader.load(ml, Predicate.class);
            long matches = sl.stream().count();
            Assertions.assertEquals(4, matches);

        }
    }
```

```
    @Nested
    @DisplayName("Confirm that SSTAFContext can be initialized from a JSONObject")
    public class JSONObjectTests {

        private JsonNode makeConfig(List<String> modulesList, List<String> pathList) {
            ObjectMapper om = new ObjectMapper();
            ObjectNode config = om.createObjectNode();

            ObjectNode moduleDef = om.createObjectNode();

            ArrayNode modules = om.createArrayNode();
            for (String module : modulesList) {
                modules.add(module);
            }

            ArrayNode paths = om.createArrayNode();
            for (String path : pathList) {
                paths.add(path);
            }

            moduleDef.set("modules", modules);
            moduleDef.set("modulePaths", paths);

            config.put("class", "mil.sstaf.core.configuration.SSTAFConfiguration");
            config.set("moduleLayerDefinition", moduleDef);

            return config;
        }

        private void check(int i) {
            SSTAFConfiguration config = SSTAFConfiguration.getInstance();
            Assertions.assertNotNull(config.getRootLayer());
        }

        @DisplayName("Check mil.sstaftest.fred")
        @Test
        public void test1A() {
            try {
                SSTAFConfiguration.from(makeConfig(List.of("mil.sstaftest.fred"),
                                List.of("dir1")),
                        SOURCE_FILE);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }

        @DisplayName("Check mil.sstaftest.barney")
        @Test
        public void test1B() {
            SSTAFConfiguration.from(makeConfig(List.of("mil.sstaftest.barney"),
                            List.of("dir2")),
                    SOURCE_FILE);
            check(1);
        }

        @DisplayName("Check mil.sstaftest.wilma")
        @Test
        public void test1C() {
            SSTAFConfiguration.from(makeConfig(List.of("mil.sstaftest.wilma"),
                            List.of("dir3")),
                    SOURCE_FILE);
            check(1);
        }

        @DisplayName("Check mil.sstaftest.betty")
        @Test
        public void test1D() {
            SSTAFConfiguration.from(makeConfig(List.of("mil.sstaftest.betty"),
```

```
                                List.of("dir4")),
                        SOURCE_FILE);
            check(1);
        }

        @DisplayName("Check fred and barney")
        @Test
        public void test2A() {
            SSTAFConfiguration.from(makeConfig(List.of("mil.sstaftest.fred", "mil.sstaftest.barne
y"),
                                List.of("dir1", "dir2")),
                        SOURCE_FILE);
            check(2);
        }

        @DisplayName("Check fred and wilma")
        @Test
        public void test2B() {
            SSTAFConfiguration.from(makeConfig(List.of("mil.sstaftest.fred", "mil.sstaftest.wilma
"),
                                List.of("dir1", "dir3")),
                        SOURCE_FILE);
            check(2);
        }

        @DisplayName("Check fred and betty")
        @Test
        public void test2C() {
            SSTAFConfiguration.from(makeConfig(List.of("mil.sstaftest.fred", "mil.sstaftest.betty
"),
                                List.of("dir1", "dir4")),
                        SOURCE_FILE);
            check(2);
        }

        @DisplayName("Check barney and wilma")
        @Test
        public void test2D() {
            SSTAFConfiguration.from(makeConfig(List.of("mil.sstaftest.barney", "mil.sstaftest.wil
ma"),
                                List.of("dir2", "dir3")),
                        SOURCE_FILE);
            check(2);
        }

        @DisplayName("Check barney and betty")
        @Test
        public void test2E() {
            SSTAFConfiguration.from(makeConfig(List.of("mil.sstaftest.barney", "mil.sstaftest.bet
ty"),
                                List.of("dir2", "dir4")),
                        SOURCE_FILE);
            check(2);
        }

        @DisplayName("Check wilma and betty")
        @Test
        public void test2F() {
            SSTAFConfiguration.from(makeConfig(List.of("mil.sstaftest.wilma", "mil.sstaftest.bett
y"),
                                List.of("dir3", "dir4")),
                        SOURCE_FILE);
            check(2);
        }

        @DisplayName("Check all")
        @Test
        public void test2G() {
```

```
            SSTAFConfiguration.from(makeConfig(List.of("mil.sstaftest.wilma", "mil.sstaftest.bett
y",
                                    "mil.sstaftest.fred", "mil.sstaftest.barney"),
                            List.of("dir1", "dir2",
                                    "dir3", "dir4")),
                    SOURCE_FILE);
            check(4);
        }

        @DisplayName("Check all - scrambled dirs")
        @Test
        public void test2H() {
            SSTAFConfiguration.from(makeConfig(List.of("mil.sstaftest.wilma", "mil.sstaftest.bett
y",
                                    "mil.sstaftest.fred", "mil.sstaftest.barney"),
                            List.of("dir4", "dir3",
                                    "dir2", "dir1")),
                    SOURCE_FILE);
            check(4);
        }

        @DisplayName("Check all - scrambled dirs, absolute paths")
        @Test
        public void test2I() {
            List<String> relatives = List.of("src/test/resources/moduleLayers/dir4", "src/test/re
sources/moduleLayers/dir3",
                    "src/test/resources/moduleLayers/dir2", "src/test/resources/moduleLayers/dir1
");
            String cwd = System.getProperty("user.dir");
            List<String> absolutes = new ArrayList<>();
            for (String s : relatives) {
                absolutes.add(cwd + File.separator + s);
            }
            SSTAFConfiguration.from(makeConfig(List.of("mil.sstaftest.wilma", "mil.sstaftest.bett
y",
                                    "mil.sstaftest.fred", "mil.sstaftest.barney"), absolutes),
                    SOURCE_FILE);
            check(4);
        }

    }
}
```

A.284 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/configuration/SSTAFConfigurationToo.java

```
package mil.sstaf.core.configuration;

import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.Getter;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import mil.sstaf.core.configuration.SSTAFConfiguration;

/**
 * Test Subclass of SSTAFConfiguration
 */
@SuperBuilder
@Jacksonized
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
public class SSTAFConfigurationToo extends SSTAFConfiguration {

    @Getter
    private String more;

}
```

A.285 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/entity/AddressTest.java

```
package mil.sstaf.core.entity;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;
import java.util.List;
import static org.junit.jupiter.api.Assertions.*;
public class AddressTest {
    @Test
    void canCreateAFullAddress() {
        EntityHandle eh = TestEntity.builder().build().getHandle();
        String handlerName = "Bob";
        Address address = Address.makeAddress(eh, handlerName);
        assertEquals(eh, address.entityHandle);
        assertEquals(handlerName, address.handlerName);
        Assertions.assertTrue(address.isExternal());
        Assertions.assertFalse(address.isInternal());
    }
    @Test
    void equalsAndHashcodeWork() {
        EntityHandle eh1 = TestEntity.builder().build().getHandle();
        String handlerName1 = "Bob";
        Address address1a = Address.makeAddress(eh1, handlerName1);
        Address address1b = Address.makeAddress(eh1, handlerName1);
        EntityHandle eh2 = TestEntity.builder().build().getHandle();
        String handlerName2 = "Jackie";
        Address address2a = Address.makeAddress(eh2, handlerName2);
        assertEquals(address1a, address1b);
        assertNotEquals(address1a, address2a);
        assertEquals(address1a.hashCode(), address1b.hashCode());
        assertNotEquals(address1a.hashCode(), address2a.hashCode());
    }
    @Test
    void canCreateAnExternalAddress() {
        EntityHandle eh = TestEntity.builder().build().getHandle();
        Address address = Address.makeExternalAddress(eh);
        assertEquals(eh, address.entityHandle);
        assertNull(address.handlerName);
        Assertions.assertTrue(address.isExternal());
        Assertions.assertFalse(address.isInternal());
    }
    @Test
    void canCreateAnInternalAddress() {
        String handlerName = "Bob";
        Address address = Address.makeInternalAddress(handlerName);
        assertNull(address.entityHandle);
        assertEquals(handlerName, address.handlerName);
        Assertions.assertFalse(address.isExternal());
        Assertions.assertTrue(address.isInternal());
    }
    @Test
    void addressComparatorWorks() {
        List<Entity> entities = List.of(TestEntity.builder().build(),
                TestEntity.builder().build(),
                TestEntity.builder().build(),
                TestEntity.builder().build());
        entities.forEach(entity -> entity.setForce(Force.BLUE));
        List<String> handlers = List.of("Handler1", "Handler2", "Handler3", "Handler4");
        Address[][] addresses = new Address[4][4];
        for (int i = 0; i < 4; ++i) {
            for (int j = 0; j < 4; ++j) {
                addresses[i][j] = Address.makeAddress(entities.get(i).getHandle(), handlers.get(j
));
            }
```

```
        }
        for (int i = 0; i < 4; ++i) {
            for (int j = 0; j < 4; ++j) {
                for (int k = 0; k < 4; ++k) {
                    for (int l = 0; l < 4; ++l) {
                        Address first = addresses[i][j];
                        Address second = addresses[k][l];
                        int result = Address.COMPARATOR.compare(first, second);
                        //logger.debug("{} {} {} {} = {} ", i,j,k,l,result);
                        if (i == k) {
                            if (j == l) {
                                Assertions.assertEquals(0, result);
                            } else if (j < l) {
                                Assertions.assertEquals(-1, result);
                            } else {
                                Assertions.assertEquals(1, result);
                            }
                        }
                    }
                }
            }
        }
        for (int i = 0; i < 4; ++i) {
            for (int j = 0; j < 4; ++j) {
                Address first = addresses[i][j];
                Address external = Address.makeExternalAddress(first.entityHandle);
                Address internal = Address.makeInternalAddress(first.handlerName);
                Assertions.assertEquals(-1, Address.COMPARATOR.compare(first, external));
                Assertions.assertEquals(1, Address.COMPARATOR.compare(first, internal));
                Assertions.assertEquals(1, Address.COMPARATOR.compare(external, first));
                Assertions.assertEquals(-1, Address.COMPARATOR.compare(internal, first));
            }
        }
    }
}
```

A.286 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/entity/EntityHandleTest.java

```
package mil.sstaf.core.entity;
import lombok.experimental.SuperBuilder;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;
import static mil.sstaf.core.entity.Force.*;
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertTrue;
class EntityHandleTest {
    @Test
    void testToString() {
        EntityHandle ep1 = new EntityHandle(makeEntity("Fred"));
        ep1.setForce(BLUE);
        String string = ep1.toString();
        String answer = "id=" + ep1.getId() + " forces=BLUE path=PATH1:PATH2:Fred name=Fred";
        assertEquals(answer, string);
    }
    @Test
    void getForce() {
        EntityHandle ep1 = new EntityHandle(makeEntity("Fred"));
        EntityHandle ep2 = new EntityHandle(makeEntity("Fred"));
        EntityHandle ep3 = new EntityHandle(makeEntity("Fred"));
        ep1.setForce(BLUE);
        ep2.setForce(RED);
        ep3.setForce(GRAY);
        assertEquals(BLUE, ep1.getForce());
        assertEquals(RED, ep2.getForce());
        assertEquals(GRAY, ep3.getForce());
    }
    @Test
    void getId() {
        EntityHandle ep1 = new EntityHandle(makeEntity("Fred"));
        EntityHandle ep2 = new EntityHandle(makeEntity("Fred"));
        EntityHandle ep3 = new EntityHandle(makeEntity("Fred"));
        assertTrue(ep1.getId() > MessageDriven.BlockCounter.USER_BLOCK_BEGIN);
        assertEquals(ep1.getId() + 1, ep2.getId());
        assertEquals(ep2.getId() + 1, ep3.getId());
    }
    @Test
    void getName() {
        EntityHandle ep1 = new EntityHandle(makeEntity("Fred"));
        EntityHandle ep2 = new EntityHandle(makeEntity("Wilma"));
        EntityHandle ep3 = new EntityHandle(makeEntity("Barney"));
        assertEquals("Fred", ep1.getName());
        assertEquals("Wilma", ep2.getName());
        assertEquals("Barney", ep3.getName());
    }
    @Test
    void testEquals() {
        EntityHandle ep1 = new EntityHandle(makeEntity("Fred"));
        EntityHandle ep2 = new EntityHandle(makeEntity("Wilma"));
        EntityHandle ep3 = new EntityHandle(makeEntity("Barney"));
        assertEquals(ep1, ep1);
        assertEquals(ep2, ep2);
        assertEquals(ep3, ep3);
        Assertions.assertNotEquals(ep1, ep2);
        Assertions.assertNotEquals(ep1, ep3);
        Assertions.assertNotEquals(ep2, ep3);
        Assertions.assertNotEquals(ep3, ep1);
        Assertions.assertNotEquals(ep2, ep1);
        Assertions.assertNotEquals(ep3, ep2);
    }
    @Test
```

```
    void testHashCode() {
        BaseEntity fred = makeEntity("Fred");
        EntityHandle ep1 = new EntityHandle(fred);
        assertEquals(fred.hashCode(), ep1.hashCode());
    }
    @Test
    void compareTo() {
        EntityHandle ep1 = new EntityHandle(makeEntity("Fred"));
        EntityHandle ep2 = new EntityHandle(makeEntity("Wilma"));
        EntityHandle ep3 = new EntityHandle(makeEntity("Barney"));
        assertTrue(ep1.compareTo(ep2) < 0);
        assertTrue(ep2.compareTo(ep3) < 0);
        assertTrue(ep3.compareTo(ep1) > 0);
    }
    BaseEntity makeEntity(final String name) {
        var builder = FakeEntity.builder();
        builder.name(name);
        return builder.build();
    }
    @SuperBuilder
    static class FakeEntity extends BaseEntity {
        @Override
        public String getPath() {
            return "PATH1:PATH2:" + getName();
        }
    }
}
```

A.287 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/entity/EntityRegistryTest.java

```
package mil.sstaf.core.entity;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Nested;
import org.junit.jupiter.api.Test;

import java.util.Collection;
import java.util.List;
import java.util.Optional;
import static org.junit.jupiter.api.Assertions.*;
public class EntityRegistryTest {
    @Nested
    @DisplayName("Test the 'Happy Path'")
    class HappyTests{
        @Test
        @DisplayName("Confirm that an entity can be registered and retrieved by ID")
        void testEntityRegistry1 () {
            EntityRegistry entityRegistry = new EntityRegistry();
            DummyEntity bob = new DummyEntity("Bob", 123);
            DummyEntity fred = new DummyEntity("Fred", 456);
            entityRegistry.registerEntity(Force.BLUE, bob);
            entityRegistry.registerEntity(Force.RED, fred);
            entityRegistry.compileEntityMaps();
            assertEquals(2, entityRegistry.getAllEntities().size());
            assertTrue(entityRegistry.getEntity(123L).isPresent());
            assertTrue(entityRegistry.getEntity(456L).isPresent());
            assertFalse(entityRegistry.getEntity(314L).isPresent());
        }
        @Test
        @DisplayName("Confirm that an entity can be registered and retrieved by EntityHandle")
        void testEntityRegistry2 () {
            EntityRegistry entityRegistry = new EntityRegistry();
            DummyEntity bob = new DummyEntity("Bob", 123);
            DummyEntity fred = new DummyEntity("Fred", 456);
            entityRegistry.registerEntity(Force.BLUE, bob);
            entityRegistry.registerEntity(Force.RED, fred);
            entityRegistry.compileEntityMaps();
            assertEquals(2, entityRegistry.getAllEntities().size());
            EntityHandle eh1 = bob.getHandle();
            assertNotNull(eh1);
            EntityHandle eh2 = fred.getHandle();
            assertNotNull(eh2);
            Optional<Entity> optBob = entityRegistry.getEntityByHandle(eh1);
            Optional<Entity> optFred = entityRegistry.getEntityByHandle(eh2);
            assertTrue(optBob.isPresent());
            assertTrue(optFred.isPresent());
            assertEquals(bob, optBob.get());
            assertEquals(fred, optFred.get());
        }
        @Test
        @DisplayName("Confirm that support for the client address works")
        void testEntityRegistry3 () {
            EntityRegistry entityRegistry = new EntityRegistry();
            EntityHandle entityHandle = EntityHandle.makeDummyHandle();
            entityRegistry.setClientAddress(entityHandle);
            EntityHandle eh = entityRegistry.getClientAddress().entityHandle;
            assertEquals(entityHandle, eh);
        }
        @Test
        @DisplayName("Confirm that all entities are in the registry")
        void testEntityRegistry4 () {
            EntityRegistry entityRegistry = new EntityRegistry();
```

```
            List<DummyEntity> dummies = List.of(new DummyEntity("Bob", 123),
                    new DummyEntity("Fred", 456),
                    new DummyEntity("Wilma", 342134));
            for (BaseEntity e : dummies) {
                entityRegistry.registerEntity(Force.BLUE, e);
            }
            entityRegistry.compileEntityMaps();
            Collection<EntityHandle> allEHs = entityRegistry.getAllEntityHandles();
            Assertions.assertEquals(3, allEHs.size());
            for (Entity e : dummies) {
                Assertions.assertTrue(allEHs.contains(e.getHandle()));
            }
        }
        @Test
        @DisplayName("Confirm that all system entities are not in the simulation entities registr
y")
        void testEntityRegistry5 () {
            EntityRegistry entityRegistry = new EntityRegistry();
            List<DummyEntity> dummies = List.of(new DummyEntity("Bob", 123),
                    new DummyEntity("Fred", 456),
                    new DummyEntity("Wilma", 342134));
            for (BaseEntity e : dummies) {
                entityRegistry.registerEntity(Force.BLUE, e);
            }
            entityRegistry.compileEntityMaps();
            DummyEntity systemThing = new DummyEntity("Wacko", -1024);
            entityRegistry.registerEntity(Force.SYSTEM, systemThing);
            Collection<Entity> entities = entityRegistry.getSimulationEntities();
            Assertions.assertEquals(3, entities.size());
            for (Entity e : dummies) {
                Assertions.assertTrue(entities.contains(e));
            }
            Assertions.assertFalse(entities.contains(systemThing));
        }

        @Test
        @DisplayName("Confirm that getting an EntityHandle by ID works")
        void testEntityRegistry6 () {
            EntityRegistry entityRegistry = new EntityRegistry();
            DummyEntity bob = new DummyEntity("Bob", 123);
            DummyEntity fred = new DummyEntity("Fred", 456);
            entityRegistry.registerEntity(Force.BLUE, bob);
            entityRegistry.registerEntity(Force.RED, fred);
            entityRegistry.compileEntityMaps();
            Optional<EntityHandle> optBob = entityRegistry.getHandle(123L);
            Assertions.assertTrue(optBob.isPresent());
            Assertions.assertEquals(bob.getHandle(), optBob.get());
            Optional<EntityHandle> optFred = entityRegistry.getHandle(456L);
            Assertions.assertTrue(optFred.isPresent());
            Assertions.assertEquals(fred.getHandle(), optFred.get());
            Optional<EntityHandle> optNobody = entityRegistry.getHandle(314L);
            Assertions.assertFalse(optNobody.isPresent());
        }
    }
    static class DummyEntity extends BaseEntity {
        DummyEntity(String name, long id) {
            super(name, id);
        }
        @Override
        public String getPath() {
            return getName();
        }

    }
}
```

A.288 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/entity/EntityTest.java

```
package mil.sstaf.core.entity;
import mil.sstaf.core.features.*;
import mil.sstaf.core.util.Injector;
import org.junit.jupiter.api.Test;
import java.util.List;
import static org.junit.jupiter.api.Assertions.*;
class EntityTest {
    private static final String ERROR_MSG = "I MEANT TO DO THAT!";
    @Test
    void canCreateEntity() {
        Entity e = TestEntity.builder().build();
        assertNotNull(e);
    }
    @Test
    void entityCanAcceptMessages() {
        BaseEntity e = TestEntity.builder().build();
        e.init();
        assertNotNull(e);
        var b = EntityAction.builder();
        b.content(StringContent.of("Banana"));
        b.source(Address.makeExternalAddress(e.getHandle()));
        b.destination(Address.makeExternalAddress(e.getHandle()));
        EntityAction em = b.build();
        e.receive(em);
        assertEquals(1, e.getInboundQueueDepth());
    }
    @Test
    void entityCanAcceptEvents() {
        BaseEntity e = TestEntity.builder().build();
        e.init();
        assertNotNull(e);
        var b = EntityEvent.builder();
        b.content(StringContent.of("Banana"));
        b.eventTime_ms(10000);
        b.source(Address.makeExternalAddress(e.getHandle()));
        b.destination(Address.makeExternalAddress(e.getHandle()));
        EntityEvent em = b.build();
        e.receive(em);
        assertEquals(1, e.getInboundQueueDepth());
    }
    @Test
    void processEventsProcessesAllMessageAndOnlyTimelyEvents() {
        Entity from = TestEntity.builder().build();
        from.setForce(Force.BLUE);
        BaseEntity to = TestEntity.builder().build();
        to.setForce(Force.BLUE);
        assertNotNull(from);
        assertNotNull(to);
        to.init();
        from.init();
        for (int i = 1; i <= 10; ++i) {
            var b = EntityEvent.builder();
            b.content(StringContent.of("Banana-" + i));
            b.eventTime_ms(i * 1000);
            b.destination(Address.makeExternalAddress(to.getHandle()));
            b.source(Address.makeExternalAddress(from.getHandle()));
            b.respondTo(Address.makeExternalAddress(from.getHandle()));
            EntityEvent em = b.build();
            to.receive(em);
        }
        for (int i = 0; i < 3; ++i) {
            var b = EntityAction.builder();
```

```
            b.content(StringContent.of("Message" + i));
            b.destination(Address.makeExternalAddress(to.getHandle()));
            b.source(Address.makeExternalAddress(from.getHandle()));
            b.respondTo(Address.makeExternalAddress(from.getHandle()));
            EntityAction em = b.build();
            to.receive(em);
        }
        to.processMessages(1000);
        List<Message> out = to.takeOutbound();
        assertEquals(4, out.size());
        to.processMessages(2000);
        out = to.takeOutbound();
        assertEquals(1, out.size());
        to.processMessages(20000);
        out = to.takeOutbound();
        assertEquals(8, out.size());
    }
    @Test
    void registeringAProcessingHandlerWorks() {
        Handler ps =
                new BaseHandler("Thing", 0, 0, 0, false,
                        "it") {
                    @Override
                    public List<Class<? extends HandlerContent>> contentHandled() {
                        return List.of(StringContent.class);
                    }
                    @Override
                    public ProcessingResult process(HandlerContent arg, long scheduledTime_ms, lo
ng currentTime_ms, Address from, long id, Address respondTo) {
                        if (arg instanceof StringContent) {
                            String string = ((StringContent) arg).getMessage();
                            Message out = this.buildNormalResponse(StringContent.of(string), id,
respondTo);
                            return ProcessingResult.of(out);
                        }
                        return ProcessingResult.empty();
                    }
                };
        Entity from = TestEntity.builder().build();
        from.setForce(Force.BLUE);
        BaseEntity to = TestEntity.builder().build();
        to.setForce(Force.BLUE);
        assertNotNull(from);
        assertNotNull(to);
        Injector.inject(ps, from.getHandle());
        to.getFeatureManager().register(ps);
        to.init();
        from.init();
        var b = EntityEvent.builder();
        b.content(StringContent.of("Banana-1"));
        b.eventTime_ms(10300);
        b.destination(Address.makeExternalAddress(to.getHandle()));
        b.source(Address.makeExternalAddress(from.getHandle()));
        b.respondTo(Address.makeExternalAddress(from.getHandle()));
        EntityEvent em = b.build();
        to.receive(em);
        to.processMessages(11000);
        List<Message> out = to.takeOutbound();
        assertEquals(1, out.size());
        assertTrue(out.get(0) instanceof MessageResponse);
        MessageResponse mr = (MessageResponse) out.get(0);
        assertEquals(StringContent.of("Banana-1"), mr.getContent());
        assertEquals(em.getSequenceNumber(), mr.getMessageID());
    }
    @Test
    void ifAHandlerThrowsAnErrorIsProduced() {
        Handler ps =
                new BaseHandler("Banana", 0, 0, 0, false,
```

```
                        "") {
                    @Override
                    public List<Class<? extends HandlerContent>> contentHandled() {
                        return List.of(StringContent.class);
                    }
                    @Override
                    public ProcessingResult process(HandlerContent arg, long scheduledTime_ms, lo
ng currentTime_ms,
                                                    Address from, long id, Address respondTo) {
                        throw new RuntimeException(ERROR_MSG);
                    }
                };
        Entity from = TestEntity.builder().build();
        from.setForce(Force.BLUE);
        BaseEntity to = TestEntity.builder().build();
        to.setForce(Force.BLUE);
        assertNotNull(from);
        assertNotNull(to);
        Injector.inject(ps, from.getHandle());
        to.getFeatureManager().register(ps);
        to.init();
        from.init();
        var b = EntityEvent.builder();
        b.content(StringContent.of("Banana-1"));
        b.eventTime_ms(10300);
        b.destination(Address.makeExternalAddress(to.getHandle()));
        b.source(Address.makeExternalAddress(from.getHandle()));
        b.respondTo(Address.makeExternalAddress(from.getHandle()));
        EntityEvent em = b.build();
        to.receive(em);
        to.processMessages(11000);
        List<Message> out = to.takeOutbound();
        assertEquals(1, out.size());
        assertTrue(out.get(0) instanceof ErrorResponse);
        ErrorResponse er = (ErrorResponse) out.get(0);
        assertNotNull(er);
        assertNotNull(er.getContent());
        assertEquals(ExceptionCommand.class, er.getContent().getClass());
        ExceptionCommand exceptionContent = (ExceptionCommand) er.content;
        assertNotNull(exceptionContent.getThrown());
        assertEquals(ERROR_MSG, exceptionContent.getThrown().getMessage());
        System.out.println(er.getErrorDescription());
        assertTrue(er.getErrorDescription().contains("Error at time 11000 ms, processing"));
    }
}
```

A.289 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/entity/FeatureManagerTest.java

```
package mil.sstaf.core.entity;
import lombok.experimental.SuperBuilder;
import mil.sstaf.core.features.*;
import mil.sstaf.core.mocks.*;
import mil.sstaf.core.util.Injector;
import mil.sstaf.core.util.SSTAFException;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Nested;
import org.junit.jupiter.api.Test;
import java.util.List;
import java.util.UUID;
import static org.junit.jupiter.api.Assertions.*;
class FeatureManagerTest {
    FakeEntity fakeEntity;
    EntityHandle eh;
    @BeforeEach
    void setup() {
        fakeEntity = FakeEntity.builder().build();
        fakeEntity.setForce(Force.BLUE);
        eh = fakeEntity.getHandle();
    }
    @Test
    void canRegisterAHandlerByClass() {
        FeatureManager featureManager = fakeEntity.getFeatureManager();
        Injector.inject(featureManager, eh);
        final Handler h1 = new Handler1();
        final Handler h2 = new Handler2();
        featureManager.register(h1);
        featureManager.register(h2);
        featureManager.getHandlerForContent(IntContent.builder().intValue(3).build()).ifPresentOr
Else(s -> assertEquals(h2, s), () -> fail("Not found"));
        featureManager.getHandlerForContent(StringContent.of("string")).ifPresentOrElse(s -> asse
rtEquals(h1, s), () -> fail("Not found"));
        featureManager.getHandlerForContent(Command1.builder().build()).ifPresentOrElse(s -> asse
rtEquals(h1, s), () -> fail("Not found"));
        assertEquals(h1, featureManager.getHandler(StringContent.class, "Handler1"));
        assertEquals(h1, featureManager.getHandler(Command1.class, "Handler1"));
        assertEquals(h1, featureManager.getHandler(StringContent.class, null));
        assertEquals(h1, featureManager.getHandler(Command1.class, null));
        assertEquals(h2, featureManager.getHandler(IntContent.class, "Handler2"));
        assertEquals(h2, featureManager.getHandler(LongContent.class, "Handler2"));
        assertEquals(h2, featureManager.getHandler(IntContent.class, null));
        assertEquals(h2, featureManager.getHandler(LongContent.class, null));
        assertNull(featureManager.getHandler(UnsupportedCommand.class, "Handler1"));
        assertNull(featureManager.getHandler(UnsupportedCommand.class, "Handler2"));
        assertThrows(NullPointerException.class,
                () -> featureManager.getHandler(null, "Handler1"));
    }
    @Test
    void unregisteredClassYieldsEmptyOptional() {
        FeatureManager featureManager = fakeEntity.getFeatureManager();
        Injector.inject(featureManager, eh);
        assertTrue(featureManager.getHandlerForContent(Double.class).isEmpty());
    }
    @Test
    void canRegisterAHandlerAndDispatchContentToIt() {
        FeatureManager featureManager = fakeEntity.getFeatureManager();
        Injector.inject(featureManager, eh);
        Handler1 handler1 = new Handler1();
        featureManager.register(handler1);
        FeatureConfiguration fc = FeatureConfiguration.builder().build();
```

```
        fc.setSeed(1234);
        handler1.configure(fc);
        PinkyProvider pinkyProvider = new Pinky();
        pinkyProvider.configure(fc);
        featureManager.injectAll(pinkyProvider);
        Command1 tc1 = new Command1();
        assertFalse(tc1.done);
        featureManager.init();
        var b = EntityEvent.builder();
        b.destination(Address.makeExternalAddress(eh));
        b.source(Address.makeExternalAddress(eh));
        b.respondTo(Address.makeExternalAddress(eh));
        b.content(tc1);
        b.eventTime_ms(1);
        EntityEvent ee = b.build();
        ProcessingResult pr = featureManager.process(ee, 10);
        assertNotNull(pr);
        assertNotNull(pr.messages);
        assertEquals(1, pr.messages.size());
        assertTrue(pr.messages.get(0).getContent() instanceof Command1);
        Command1 tc2 = (Command1) pr.messages.get(0).getContent();
        assertTrue(tc2.done);
    }
    @Test
    void processingBeforeInitializationThrows() {
        FeatureManager featureManager = fakeEntity.getFeatureManager();
        Injector.inject(featureManager, eh);
        FeatureConfiguration fc = FeatureConfiguration.builder().build();
        fc.setSeed(1234);
        Handler1 handler1 = new Handler1();
        featureManager.register(handler1);
        handler1.configure(fc);
        var b = EntityEvent.builder();
        b.destination(Address.makeExternalAddress(eh));
        b.source(Address.makeExternalAddress(eh));
        b.respondTo(Address.makeExternalAddress(eh));
        b.content(new Command1());
        b.eventTime_ms(100);
        EntityEvent ee = b.build();
        assertThrows(SSTAFException.class,
                () -> featureManager.process(ee, 100));
    }
    @Test
    void processingUnregisteredContentThrows() {
        FeatureManager featureManager = fakeEntity.getFeatureManager();
        Injector.inject(featureManager, eh);
        FeatureConfiguration fc = FeatureConfiguration.builder().build();
        fc.setSeed(1234);
        Handler1 h1 = new Handler1();
        h1.configure(fc);
        Pinky pinky = new Pinky();
        pinky.configure(fc);
        Injector.inject(h1, pinky);
        featureManager.register(h1);
        featureManager.init();
        var b = EntityEvent.builder();
        b.destination(Address.makeExternalAddress(eh));
        b.source(Address.makeExternalAddress(eh));
        b.respondTo(Address.makeExternalAddress(eh));
        b.content(IntContent.builder().intValue(3).build());
        b.eventTime_ms(100);
        EntityEvent ee = b.build();
        ProcessingResult pr = featureManager.process(ee, 100);
        assertTrue(pr.messages.get(0) instanceof ErrorResponse);
    }
    @Test
    void injectionAndInitializationWorks() {
        FeatureConfiguration fc = FeatureConfiguration.builder().build();
```

```
        fc.setSeed(1234);
        FeatureManager featureManager = fakeEntity.getFeatureManager();
        Injector.inject(featureManager, eh);
        Handler1 h1 = new Handler1();
        h1.configure(fc);
        Handler2 h2 = new Handler2();
        h2.configure(fc);
        featureManager.register(h1);
        featureManager.register(h2);
        PinkyProvider pinky = new Pinky();
        pinky.configure(fc);
        BrainProvider brain = new Brain();
        brain.configure(fc);
        featureManager.register(pinky);
        featureManager.register(brain);
        String stringVal = "Jessica";
        Long posVal = 315444L;
        Long negVal = -454545L;
        UUID bob = UUID.randomUUID();
        featureManager.injectAll(stringVal, bob, pinky, brain);
        featureManager.injectNamed("+", posVal);
        featureManager.injectNamed("-", negVal);
        featureManager.init();
        assertEquals(h1.myString, stringVal);
        assertEquals(h1.posLong, posVal);
        assertEquals(h1.negLong, negVal);
        assertEquals(h1.pinky, pinky);
        assertEquals(h2.string, stringVal);
        assertEquals(h2.uuid, bob);
        assertEquals(h2.brain, brain);
        assertTrue(h2.isInitialized());
    }
    @Test
    void handlersLoadedViaRequiresCanBeAccessed() {
        Loaders.registerClass(Handler1.class);
        Loaders.registerClass(Handler2.class);
        Loaders.registerClass(Handler3.class);
        Loaders.registerClass(Pinky.class);
        Loaders.registerClass(Brain.class);
        var fakeEntityBuilder2 = FakeEntity.builder();
        FeatureSpecification fs = FeatureSpecification.builder()
                .featureName("Handler3").build();
        fakeEntityBuilder2.features(List.of(fs));
        FakeEntity fakeEntity2 = fakeEntityBuilder2.build();
        fakeEntity2.setForce(Force.BLUE);
        EntityHandle eh2 = fakeEntity2.getHandle();
        FeatureManager featureManager = fakeEntity2.getFeatureManager();
        Injector.inject(featureManager, eh2);
        featureManager.init();
        assertDoesNotThrow(() -> {
            assertEquals("Handler1", featureManager.getHandler(StringContent.class, "Handler1").g
etName());
            assertEquals("Handler1", featureManager.getHandler(Command1.class, "Handler1").getNam
e());
            assertEquals("Handler1", featureManager.getHandler(StringContent.class, null).getName
());
            assertEquals("Handler1", featureManager.getHandler(Command1.class, null).getName());
            assertEquals("Handler2", featureManager.getHandler(IntContent.class, "Handler2").getN
ame());
            assertEquals("Handler2", featureManager.getHandler(LongContent.class, "Handler2").get
Name());
            assertEquals("Handler2", featureManager.getHandler(IntContent.class, null).getName())
;
            assertEquals("Handler2", featureManager.getHandler(LongContent.class, null).getName()
);
            assertEquals("Handler3", featureManager.getHandler(BlobContent.class, null).getName()
);
        });
```

```
    }
    @SuperBuilder
    static class FakeEntity extends BaseEntity {
        @Override
        public String getPath() {
            return "PATH1:PATH2:" + getName();
        }
    }

    @DisplayName("Test the happy paths")
    @Nested
    class HappyTests {
        @Test
        void canRegisterAHandlerByClass() {
            FeatureManager featureManager = fakeEntity.getFeatureManager();
            Injector.inject(featureManager, eh);
            final Handler h1 = new Handler1();
            final Handler h2 = new Handler2();
            featureManager.register(h1);
            featureManager.register(h2);
            featureManager.getHandlerForContent(IntContent.builder().build()).ifPresentOrElse(s -
> assertEquals(h2, s), () -> fail("Not found"));
            featureManager.getHandlerForContent(StringContent.of("string")).ifPresentOrElse(s ->
assertEquals(h1, s), () -> fail("Not found"));
            featureManager.getHandlerForContent(new Command1()).ifPresentOrElse(s -> assertEquals
(h1, s), () -> fail("Not found"));
            assertEquals(h1, featureManager.getHandler(StringContent.class, "Handler1"));
            assertEquals(h1, featureManager.getHandler(Command1.class, "Handler1"));
            assertEquals(h1, featureManager.getHandler(StringContent.class, null));
            assertEquals(h1, featureManager.getHandler(Command1.class, null));
            assertEquals(h2, featureManager.getHandler(IntContent.class, "Handler2"));
            assertEquals(h2, featureManager.getHandler(LongContent.class, "Handler2"));
            assertEquals(h2, featureManager.getHandler(IntContent.class, null));
            assertEquals(h2, featureManager.getHandler(LongContent.class, null));
            assertNull(featureManager.getHandler(UnsupportedCommand.class, "Handler1"));
            assertNull(featureManager.getHandler(UnsupportedCommand.class, "Handler2"));
            assertThrows(NullPointerException.class,
                    () -> featureManager.getHandler(null, "Handler1"));
        }
        @Test
        void unregisteredClassYieldsEmptyOptional() {
            FeatureManager featureManager = fakeEntity.getFeatureManager();
            Injector.inject(featureManager, eh);
            assertTrue(featureManager.getHandlerForContent(Double.class).isEmpty());
        }
        @Test
        void canRegisterAHandlerAndDispatchContentToIt() {
            FeatureConfiguration fc = FeatureConfiguration.builder().build();
            fc.setSeed(1234);
            FeatureManager featureManager = fakeEntity.getFeatureManager();
            Injector.inject(featureManager, eh);
            Handler1 handler1 = new Handler1();
            featureManager.register(handler1);
            handler1.configure(fc);
            PinkyProvider pinkyProvider = new Pinky();
            pinkyProvider.configure(fc);
            featureManager.injectAll(pinkyProvider);
            Command1 tc1 = new Command1();
            assertFalse(tc1.done);
            featureManager.init();
            var b = EntityEvent.builder();
            b.destination(Address.makeExternalAddress(eh));
            b.source(Address.makeExternalAddress(eh));
            b.respondTo(Address.makeExternalAddress(eh));
            b.content(tc1);
            b.eventTime_ms(1);
            EntityEvent ee = b.build();
            ProcessingResult pr = featureManager.process(ee, 10);
```

```
            assertNotNull(pr);
            assertNotNull(pr.messages);
            assertEquals(1, pr.messages.size());
            assertTrue(pr.messages.get(0).getContent() instanceof Command1);
            Command1 tc2 = (Command1) pr.messages.get(0).getContent();
            assertTrue(tc2.done);
        }
        @Test
        void injectionAndInitializationWorks() {
            FeatureConfiguration fc = FeatureConfiguration.builder().build();
            fc.setSeed(1234);
            FeatureManager featureManager = fakeEntity.getFeatureManager();
            Injector.inject(featureManager, eh);
            Handler1 h1 = new Handler1();
            h1.configure(fc);
            Handler2 h2 = new Handler2();
            h2.configure(fc);
            featureManager.register(h1);
            featureManager.register(h2);
            PinkyProvider pinky = new Pinky();
            pinky.configure(fc);
            BrainProvider brain = new Brain();
            brain.configure(fc);
            featureManager.register(pinky);
            featureManager.register(brain);
            String stringVal = "Jessica";
            Long posVal = 315444L;
            Long negVal = -454545L;
            UUID bob = UUID.randomUUID();
            featureManager.injectAll(stringVal, bob, pinky, brain);
            featureManager.injectNamed("+", posVal);
            featureManager.injectNamed("-", negVal);
            featureManager.init();
            assertEquals(h1.myString, stringVal);
            assertEquals(h1.posLong, posVal);
            assertEquals(h1.negLong, negVal);
            assertEquals(h1.pinky, pinky);
            assertEquals(h2.string, stringVal);
            assertEquals(h2.uuid, bob);
            assertEquals(h2.brain, brain);
            assertTrue(h2.isInitialized());
        }
        @Test
        void handlersLoadedViaRequiresCanBeAccessed() {
            Loaders.registerClass(Handler1.class);
            Loaders.registerClass(Handler2.class);
            Loaders.registerClass(Handler3.class);
            Loaders.registerClass(Pinky.class);
            Loaders.registerClass(Brain.class);
            var fakeEntityBuilder2 = FakeEntity.builder();
            FeatureSpecification fs = FeatureSpecification.builder()
                    .featureName("Handler3").build();
            fakeEntityBuilder2.feature(fs);
            FakeEntity fakeEntity2 = fakeEntityBuilder2.build();
            fakeEntity2.setForce(Force.BLUE);
            EntityHandle eh2 = fakeEntity2.getHandle();
            FeatureManager featureManager = fakeEntity2.getFeatureManager();
            Injector.inject(featureManager, eh2);
            featureManager.init();
            assertDoesNotThrow(() -> {
                assertEquals("Handler1", featureManager.getHandler(StringContent.class, "Handler1
").getName());
                assertEquals("Handler1", featureManager.getHandler(Command1.class, "Handler1").ge
tName());
                assertEquals("Handler1", featureManager.getHandler(StringContent.class, null).get
Name());
                assertEquals("Handler1", featureManager.getHandler(Command1.class, null).getName(
));
```

```
                assertEquals("Handler2", featureManager.getHandler(IntContent.class, "Handler2").
getName());
                assertEquals("Handler2", featureManager.getHandler(LongContent.class, "Handler2")
.getName());
                assertEquals("Handler2", featureManager.getHandler(IntContent.class, null).getNam
e());
                assertEquals("Handler2", featureManager.getHandler(LongContent.class, null).getNa
me());
                assertEquals("Handler3", featureManager.getHandler(BlobContent.class, null).getNa
me());
            });
        }
    }
    @DisplayName("Test the failure modes")
    @Nested
    class FailureTests {
        @Test
        void processingBeforeInitializationThrows() {
            FeatureConfiguration fc = FeatureConfiguration.builder().build();
            fc.setSeed(1234);
            FeatureManager featureManager = fakeEntity.getFeatureManager();
            Injector.inject(featureManager, eh);
            Handler1 handler1 = new Handler1();
            featureManager.register(handler1);
            handler1.configure(fc);
            var b = EntityEvent.builder();
            b.destination(Address.makeExternalAddress(eh));
            b.source(Address.makeExternalAddress(eh));
            b.respondTo(Address.makeExternalAddress(eh));
            b.content(new Command1());
            b.eventTime_ms(100);
            EntityEvent ee = b.build();
            assertThrows(SSTAFException.class,
                    () -> featureManager.process(ee, 100));
        }
        @Test
        void processingUnregisteredContentThrows() {
            FeatureConfiguration fc = FeatureConfiguration.builder().build();
            fc.setSeed(1234);
            FeatureManager featureManager = fakeEntity.getFeatureManager();
            Injector.inject(featureManager, eh);
            Handler1 h1 = new Handler1();
            h1.configure(fc);
            Pinky pinky = new Pinky();
            pinky.configure(fc);
            Injector.inject(h1, pinky);
            featureManager.register(h1);
            featureManager.init();
            var b = EntityEvent.builder();
            b.destination(Address.makeExternalAddress(eh));
            b.source(Address.makeExternalAddress(eh));
            b.respondTo(Address.makeExternalAddress(eh));
            b.content(IntContent.builder().intValue(3).build());
            b.eventTime_ms(100);
            EntityEvent ee = b.build();
            ProcessingResult pr = featureManager.process(ee, 100);
            assertTrue(pr.messages.get(0) instanceof ErrorResponse);
        }
    }
}
```

A.290 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/entity/HumanFactoryTest.java

```
package mil.sstaf.core.entity;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.node.ObjectNode;
import mil.sstaf.core.util.SSTAFException;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Nested;
import org.junit.jupiter.api.Test;
import java.io.File;
import java.nio.file.Path;
import static org.junit.jupiter.api.Assertions.*;
class HumanFactoryTest {
    static Path USER_DIR = Path.of(System.getProperty("user.dir"));
    static Path RESOURCE_DIR = Path.of(USER_DIR.toString(), "src/test/resources/HumanFactoryTest"
);
    private File makeFile(String filename) {
        Path path = Path.of(RESOURCE_DIR.toString(), filename);
        return path.toFile();
    }
    @Nested
    @DisplayName("Test the 'Happy Path'")
    class HappyTests {
        @Test
        @DisplayName("Confirm that a configuration file that consists only of references can be l
oaded correctly")
        void allReferenceJSONFileWorks() {
            Assertions.assertDoesNotThrow(() -> {
                String filename = "TestHuman1.json";
                File f = makeFile(filename);
                Human human = Human.from(f);
                assertNotNull(human);
            });
        }
        @Test
        @DisplayName("Confirm that a good JSON configuration file can be loaded")
        void goodJSONWorks() {
            ObjectMapper om = new ObjectMapper();
            ObjectNode jsonObject = om.createObjectNode();
            jsonObject.put("class", "mil.sstaf.core.entity.Human");
            jsonObject.put("name", "Test Dude");
            ObjectNode procNode = om.createObjectNode();
            jsonObject.set("configurations", procNode);
            jsonObject.set("features", om.createArrayNode());
            Human human = Human.from(jsonObject, RESOURCE_DIR);
            assertNotNull(human);
        }

    }
    @Nested
    @DisplayName("Test failure modes")
    class FailureTests {
        @Test
        @DisplayName("Confirm that a attempting to load a malformed file will throw a SSTAFExcept
ion")
        void badJSONThrows() {
            assertThrows(SSTAFException.class, () -> {
                ObjectMapper om = new ObjectMapper();
                ObjectNode jsonObject = om.createObjectNode();
                Human human = Human.from(jsonObject, RESOURCE_DIR);
                assertNotNull(human);
            });
        }
```

```
        @Test
        void missingFileThrows() {
            assertThrows(SSTAFException.class, () -> {
                        String filename = "NotThere.json";
                        File file = makeFile(filename);
                        Human human = Human.from(file);
                        assertNull(human);
                    }
            );
        }
        @Test
        void badFileThrows() {
            assertThrows(SSTAFException.class, () -> {
                        String filename = "BadHuman1.json";
                        File file = makeFile(filename);
                        Human human = Human.from(file);
                        assertNull(human);
                    }
            );
        }
    }
}
```

A.291 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/entity/HumanTest.java

```
package mil.sstaf.core.entity;
import mil.sstaf.core.features.BaseFeature;
import mil.sstaf.core.features.Feature;
import mil.sstaf.core.features.FeatureConfiguration;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
class HumanTest {
    private static final String uuidPattern =
            "([a-fA-F0-9]{8}-[a-fA-F0-9]{4}-[a-fA-F0-9]{4}-[a-fA-F0-9]{4}-[a-fA-F0-9]{12})";

    @Test
    void settingNameAndToStringWork() {
        var person = Human.builder().build();
        person.setName("Leroy Brown");
        String s = person.toString();
        assertTrue(s.contains("Leroy Brown"));
        assertTrue(s.contains("definitionName='UNSPECIFIED'"));
    }
    @Test
    void namingBehaviorWorks() {
        Human h1 = Human.builder().build();
        assertTrue(h1.getName().matches(uuidPattern), "Name was not a UUID");
        assertEquals("UNSPECIFIED", h1.getDefinitionName(), "Definition names did not match");
        var hb = Human.builder();
        hb.definitionName("Bob");
        Human h2 = hb.build();
        assertTrue(h2.getName().matches(uuidPattern));
        assertEquals("Bob", h2.getDefinitionName(), "Definition names did not match");
        hb = Human.builder();
        hb.definitionName("Super-Soldier");
        hb.name("Steve Rogers");
        Human h3 = hb.build();
        assertEquals("Super-Soldier", h3.getDefinitionName());
        assertEquals("Steve Rogers", h3.getName());
    }
    @Test
    void canSpecifyProvidersAndInitialize() {
        Human h1 = Human.builder().build();
        h1.setForce(Force.GRAY);
        FeatureManager featureManager = h1.getFeatureManager();
        featureManager.register(new StringThing());
        featureManager.register(new Bananarama());
        Assertions.assertDoesNotThrow(h1::init);
    }
    @Test
    void equalityAndHash() {
        Human person = Human.builder().build();
        Human anotherPerson = Human.builder().build();
        assertEquals(person, person);
        assertEquals(anotherPerson, anotherPerson);
        assertNotEquals(person, anotherPerson);
        assertNotEquals(person.hashCode(), anotherPerson.hashCode());
        final class Junk {
        }
        Junk notAPerson = new Junk();
        assertNotEquals(person, notAPerson);
        Human.HumanBuilder<?, ?> hb = Human.builder();
        hb.name("Pat");
        Human personNamedPat = hb.build();
        assertNotEquals(person, personNamedPat);
    }
```

```
    interface StringProvider extends Feature {
    }
    interface BananaProvider extends Feature {
    }
    static class StringThing extends BaseFeature implements StringProvider {
        public StringThing() {
            super("Senator Vrenak", 3, 1, 4, false, "It's a fake!!!!");
        }

        @Override
        public Class<? extends FeatureConfiguration> getConfigurationClass() {
            return FeatureConfiguration.class;
        }
    }
    static class Bananarama extends BaseFeature implements BananaProvider {
        public Bananarama() {
            super("Bananarama", 80, 81, 82, false, "Pop!");
        }
        @Override
        public Class<? extends FeatureConfiguration> getConfigurationClass() {
            return FeatureConfiguration.class;
        }
    }
}
```

A.292 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/entity/MessageQueueComparatorTest.java

```
package mil.sstaf.core.entity;
import mil.sstaf.core.features.ExceptionCommand;
import mil.sstaf.core.features.StringContent;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertEquals;
class MessageQueueComparatorTest {
    @Test
    void errorResponsesOrderedCorrectly() {
        MessageQueueComparator comp = new MessageQueueComparator();
        Entity testEntity1 = TestEntity.builder().build();
        Entity testEntity2 = TestEntity.builder().build();
        testEntity1.setForce(Force.BLUE);
        testEntity2.setForce(Force.BLUE);
        var b1 = ErrorResponse.builder()
                .content(ExceptionCommand.builder().thrown(new Throwable()).build())
                .source(Address.makeExternalAddress(testEntity1.getHandle()))
                .destination(Address.makeExternalAddress(testEntity2.getHandle()));
        var err1 = b1.build();
        var b2 = ErrorResponse.builder()
                .source(Address.makeExternalAddress(testEntity2.getHandle()))
                .destination(Address.makeExternalAddress(testEntity1.getHandle()))
                .content(ExceptionCommand.builder().thrown(new Throwable()).build());
        var err2 = b2.build();
        assertEquals(-1, comp.compare(err1, err2));
    }
    @Test
    @SuppressWarnings(value="all")
    void entityEventsOrderedCorrectly() {
        MessageQueueComparator comp = new MessageQueueComparator();
        Entity testEntity1 = TestEntity.builder().build();
        Entity testEntity2 = TestEntity.builder().build();
        testEntity1.setForce(Force.BLUE);
        testEntity2.setForce(Force.BLUE);
        var b1 = EntityEvent.builder()
                .eventTime_ms(10300)
                .source(Address.makeExternalAddress(testEntity1.getHandle()))
                .destination(Address.makeExternalAddress(testEntity2.getHandle()))
                .content(StringContent.builder().message("").build());
        EntityEvent entityEvent1 = b1.build();
        var b2 = EntityEvent.builder();
        b2.eventTime_ms(15000);
        b2.source(Address.makeExternalAddress(testEntity1.getHandle()));
        b2.destination(Address.makeExternalAddress(testEntity2.getHandle()));
        b2.content(StringContent.builder().message("").build());
        EntityEvent entityEvent2 = b2.build();
        var b3 = EntityEvent.builder();
        b3.eventTime_ms(10300);
        b3.source(Address.makeExternalAddress(testEntity2.getHandle()));
        b3.destination(Address.makeExternalAddress(testEntity1.getHandle()));
        b3.content(StringContent.builder().message("'").build());
        EntityEvent entityEvent3 = b3.build();
        assertEquals(-1, comp.compare(entityEvent1, entityEvent2));
        assertEquals(-1, comp.compare(entityEvent1, entityEvent3));
        assertEquals(0, comp.compare(entityEvent1, entityEvent1));
        assertEquals(1, comp.compare(entityEvent2, entityEvent1));
        assertEquals(1, comp.compare(entityEvent3, entityEvent1));
    }
    @Test
    void eventsAndMessagesOrderedCorrectly() {
        MessageQueueComparator comp = new MessageQueueComparator();
        Entity testEntity1 = TestEntity.builder().build();
        Entity testEntity2 = TestEntity.builder().build();
```

```
        testEntity1.setForce(Force.BLUE);
        testEntity2.setForce(Force.BLUE);
        var b1 = ErrorResponse.builder()
                .content(ExceptionCommand.builder().thrown(new Throwable()).build())
                .source(Address.makeExternalAddress(testEntity1.getHandle()))
                .destination(Address.makeExternalAddress(testEntity2.getHandle()));
        ErrorResponse err1 = b1.build();
        var b2 = ErrorResponse.builder()
                .content(ExceptionCommand.builder().thrown(new Throwable()).build())
                .source(Address.makeExternalAddress(testEntity2.getHandle()))
                .destination(Address.makeExternalAddress(testEntity1.getHandle()));
        ErrorResponse err2 = b2.build();
        var b3 = EntityEvent.builder()
                .eventTime_ms(0)
                .source(Address.makeExternalAddress(testEntity1.getHandle()))
                .destination(Address.makeExternalAddress(testEntity2.getHandle()))
                .content(StringContent.builder().message("").build());
        EntityEvent entityEvent1 = b3.build();
        var b4 = EntityEvent.builder()
                .eventTime_ms(5000)
                .source(Address.makeExternalAddress(testEntity1.getHandle()))
                .destination(Address.makeExternalAddress(testEntity2.getHandle()))
                .content(StringContent.builder().message("").build());
        EntityEvent entityEvent2 = b4.build();
        var b5 = EntityEvent.builder()
                .eventTime_ms(10300)
                .source(Address.makeExternalAddress(testEntity2.getHandle()))
                .destination(Address.makeExternalAddress(testEntity1.getHandle()))
                .content(StringContent.builder().message("").build());
        EntityEvent entityEvent3 = b5.build();
        assertEquals(1, comp.compare(entityEvent1, err1));
        assertEquals(1, comp.compare(entityEvent2, err1));
        assertEquals(1, comp.compare(entityEvent3, err1));
        assertEquals(1, comp.compare(entityEvent1, err2));
        assertEquals(1, comp.compare(entityEvent2, err2));
        assertEquals(1, comp.compare(entityEvent3, err2));
        assertEquals(-1, comp.compare(err1, entityEvent1));
        assertEquals(-1, comp.compare(err1, entityEvent2));
        assertEquals(-1, comp.compare(err1, entityEvent3));
        assertEquals(-1, comp.compare(err2, entityEvent1));
        assertEquals(-1, comp.compare(err2, entityEvent2));
        assertEquals(-1, comp.compare(err2, entityEvent3));
        assertEquals(-1, comp.compare(err2, entityEvent3));
    }
}
```

A.293 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/entity/RankTest.java

```
package mil.sstaf.core.entity;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertNull;
public class RankTest {
    @Test
    void testRankMatching() {
        for (Rank rank : Rank.values()) {
            assertEquals(rank, Rank.findMatch(rank.name()));
            assertEquals(rank, Rank.findMatch(rank.getCode()));
        }
        assertNull(Rank.findMatch("ADM")); // No Navy!!
        assertNull(Rank.findMatch("A1C")); // No Air Force!!
        assertNull(Rank.findMatch(null)); // No nothing!!
    }
}
```

A.294 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/entity/SoldierFactoryTest.java

```
package mil.sstaf.core.entity;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.node.ObjectNode;
import mil.sstaf.core.util.SSTAFException;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Nested;
import org.junit.jupiter.api.Test;
import java.io.File;
import java.nio.file.Path;
import static org.junit.jupiter.api.Assertions.*;
class SoldierFactoryTest {

    static Path USER_DIR = Path.of(System.getProperty("user.dir"));
    static Path RESOURCE_DIR = Path.of(USER_DIR.toString(), "src/test/resources/SoldierFactoryTes
t");
    private File makeFile(String filename) {
        Path path = Path.of(RESOURCE_DIR.toString(), filename);
        return path.toFile();
    }

    @Nested
    @DisplayName("Test the Happy Path where everything works")
    class HappyTests {
        @Test
        @DisplayName("Confirm that a file that uses only references resolves correctly")
        void allReferenceJSONFileWorks() {
            String filename = "TestSoldier1.json";
            Soldier soldier = Soldier.from(makeFile(filename));
            assertNotNull(soldier);
            assertEquals(Rank.E6, soldier.getRank());
        }
        @Test
        @DisplayName("Confirm that good JSON works")
        void goodJSONWorks() {
            ObjectMapper om = new ObjectMapper();
            ObjectNode soldier1 = om.createObjectNode();
            soldier1.put("class", "mil.sstaf.core.entity.Soldier");
            soldier1.put("name", "Test Dude");
            soldier1.put("rank", "E6");

            soldier1.set("features", om.createArrayNode());
            soldier1.set("configurations", om.createObjectNode());
            Soldier soldier = Soldier.from(soldier1, Path.of(RESOURCE_DIR.toString(), "soldier.js
on"));
            assertNotNull(soldier);
            assertEquals(Rank.E6, soldier.getRank());
        }
    }
    @Nested
    @DisplayName("Check the failure modes")
    class UnhappyTests {
        @Test
        @DisplayName("Confirm that a file with bad JSON throws")
        void badJSONThrows() {
            assertThrows(SSTAFException.class, () -> {
                ObjectMapper om = new ObjectMapper();
                Soldier soldier = Soldier.from(om.createObjectNode(), Path.of(RESOURCE_DIR.toStri
ng(), "soldier.json"));
                assertNotNull(soldier);
            });
        }
        @Test
```

```
        @DisplayName("Confirm that a missing file throws")
        void missingFileThrows() {
            assertThrows(SSTAFException.class, () -> {
                        String filename = "NotThere.json";
                        Soldier soldier = Soldier.from(makeFile(filename));
                        assertNull(soldier);
                    }
            );
        }
        @Test
        @DisplayName("Confirm that a file with missing stuff throws")
        void badFileThrows() {
            assertThrows(SSTAFException.class, () -> {
                        String filename = "BadSoldier1.json";
                        Soldier soldier = Soldier.from(makeFile(filename));
                        assertNull(soldier);
                    }
            );
        }
        @Test
        @DisplayName("Confirm that a file with missing Antho throws")
        void missingAnthroThrows() {
            assertThrows(SSTAFException.class, () -> {
                        String filename = "BadSoldier2.json";
                        Soldier soldier = Soldier.from(makeFile(filename));
                        assertNull(soldier);
                    }
            );
        }
    }
}
```

A.295 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/entity/SoldierTest.java

```
package mil.sstaf.core.entity;

import mil.sstaf.core.features.BaseFeature;
import mil.sstaf.core.features.Feature;
import mil.sstaf.core.features.FeatureConfiguration;
import org.junit.jupiter.api.Test;
import static mil.sstaf.core.entity.Rank.E7;
import static org.junit.jupiter.api.Assertions.*;
class SoldierTest {
    private final Soldier defaultSoldier = Soldier.builder().build();
    @Test
    void emptyBuildYieldsDefaults() {
        var s = Soldier.builder().build();
        assertEquals(Rank.E1, s.getRank());
    }

    @Test
    void equalityTests() {
        Soldier anotherDefaultSoldier = Soldier.builder().build();
        assertEquals(defaultSoldier, defaultSoldier);
        assertNotEquals(defaultSoldier, anotherDefaultSoldier); // UUIDs are different
        assertNotEquals(defaultSoldier.hashCode(), anotherDefaultSoldier.hashCode()); // ditto
        var sb = Soldier.builder();
        sb.rank(Rank.E3.getCode());
        Soldier officer = sb.build();
        sb.rank(Rank.E6.getCode());
        Soldier oldSoldier = sb.build();
        sb = Soldier.builder();
        sb.name("Joe");
        Soldier giJoe = sb.build();
        // different soldiers are not equal to the default soldier
        assertNotEquals(defaultSoldier, oldSoldier, "Failed to check ages"); // a changed anthro
value (age)
        assertNotEquals(defaultSoldier, officer, "Failed to check ranks"); // changes soldier val
ue (rank)
        assertNotEquals(defaultSoldier, giJoe, "Failed to check names"); // a changed human value
(name)
    }
    @Test
    void soldierBuildReturnsASoldier() {
        assertEquals(Soldier.class, Soldier.builder().build().getClass());
    }
    @Test
    void namingWorks() {
        var  sb = Soldier.builder();
        sb.name("Bob");
        Soldier s1 = sb.build();
        var ub1 = Unit.builder();
        ub1.name("Unit 1");
        ub1.soldier(Unit.MemberSoldier.of("Commander", s1));
        Unit u1 = ub1.build();
        var ub2 = Unit.builder();
        ub2.name("Unit 2");
        Unit u2 = ub2.build();
        var ub3 = Unit.builder();
        ub3.name("Unit 3");
        Unit u3 = ub3.build();
        var ub4 = Unit.builder();
        ub4.name("Unit 4");
        Unit u4 = ub4.build();
        u1.attach(u2, u1.getName());
        u2.attach(u3, u2.getName());
```

```
        u3.attach(u4, u3.getName());
        assertEquals("Commander", s1.getPosition());
        String path = s1.getPath();
        assertEquals("Unit 4:Unit 3:Unit 2:Unit 1:Commander", path);

        var sb1 = Soldier.builder();
        sb1.name("Bob");
        s1 = sb1.build();
        ub1 = Unit.builder();
        ub1.name("Unit 1");
        u1 = ub1.build();
        ub2 = Unit.builder();
        ub2.name("Unit 2");
        u2 = ub2.build();
        ub3 = Unit.builder();
        ub3.name("Unit 3");
        u3 = ub3.build();
        ub4 = Unit.builder();
        ub4.name("Unit 4");
        u4 = ub4.build();
        u1.attach(u2, u1.getName());
        u2.attach(u3, u2.getName());
        u3.attach(u4, u3.getName());
        u1.addSoldier(s1);
        assertEquals(u1, s1.getUnit());
        assertEquals("Unit 4:Unit 3:Unit 2:Unit 1:Bob", s1.getPath());
    }
    @Test
    void canSpecifyProvidersAndInitialize() {
        HumanTest.StringThing ip = new HumanTest.StringThing();
        HumanTest.Bananarama kp = new HumanTest.Bananarama();
        ArcticCircle ap = new ArcticCircle();
        FiveGuys tp = new FiveGuys();
        var sb1 = Soldier.builder();
        Soldier s1 = sb1.build();
        s1.setForce(Force.BLUE);
        FeatureManager featureManager = s1.getFeatureManager();
        featureManager.register(ip);
        featureManager.register(kp);
        featureManager.register(ap);
        featureManager.register(tp);
        s1.init();
        assertTrue(ip.isInitialized());
        assertTrue(kp.isInitialized());
        assertTrue(ap.isInitialized());
        assertTrue(tp.isInitialized());
    }
    @Test
    void otherStuff() {
        var sb = Soldier.builder();
        sb.name("Fred");
        sb.rank(E7.getCode());
        Soldier s = sb.build();
        assertTrue(s.toString().contains("Fred"));
        assertEquals(Rank.E7, s.getRank());
    }
    interface MilkshakeProvider extends Feature {
    }
    interface BurgerProvider extends Feature {
    }

    static class ArcticCircle extends BaseFeature implements MilkshakeProvider {
        public ArcticCircle() {
            super("Arctic Circle", 0, 0, 0, false, "Churchville");
        }
        @Override
        public Class<? extends FeatureConfiguration> getConfigurationClass() {
            return FeatureConfiguration.class;
```

```
        }
    }
    static class FiveGuys extends BaseFeature implements BurgerProvider {
        public FiveGuys() {
            super("Five Guys", 0, 0, 5, false, "Rather expensive");
        }
        @Override
        public Class<? extends FeatureConfiguration> getConfigurationClass() {
            return FeatureConfiguration.class;
        }
    }
}
```

A.296 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/entity/TestEntity.java

```
package mil.sstaf.core.entity;
import lombok.experimental.SuperBuilder;
//
// Entity is abstract, ergo the only way to test it is
// to subclass it.
//
@SuperBuilder
public class TestEntity extends BaseEntity {
    public static Entity makeTestEntity() {
        return TestEntity.builder().build();
    }
    @Override
    public String getPath() {
        return null;
    }

}
```

A.297 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/entity/UnitFactoryTest.java

```
package mil.sstaf.core.entity;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.node.ArrayNode;
import com.fasterxml.jackson.databind.node.ObjectNode;
import mil.sstaf.core.json.JsonLoader;
import mil.sstaf.core.util.SSTAFException;
import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;
import java.io.File;
import java.nio.file.Path;
import static org.junit.jupiter.api.Assertions.*;
class UnitFactoryTest {

    static Path USER_DIR = Path.of(System.getProperty("user.dir"));
    static Path RESOURCE_DIR = Path.of(USER_DIR.toString(), "src/test/resources/UnitFactoryTest")
;
    private File makeFile(String filename) {
        Path path = Path.of(RESOURCE_DIR.toString(), filename);
        return path.toFile();
    }
    @Test
    void aUnitWithJustSoldiersWorks() {
        String filename = "FireTeam.json";
        Unit unit = Unit.from(makeFile(filename));
        assertNotNull(unit);
        assertEquals(4, unit.getAllMembers().size());
    }
    @Test
    void aUnitHierarchyWorks() {
        String filename = "Squad.json";
        Unit unit = Unit.from(makeFile(filename));
        assertNotNull(unit);
        assertEquals(9, unit.getAllMembers().size());
    }
    @Test
    void canBuildACompany() {
        String filename = "Company.json";
        Unit unit = Unit.from(makeFile(filename));
        assertNotNull(unit);
        assertEquals(90, unit.getAllMembers().size());
    }
    @Test
    void goodJSONWorks() {
        ObjectMapper om = new ObjectMapper();
        ArrayNode soldierList = om.createArrayNode();
        ObjectNode soldier1 = om.createObjectNode();
        soldier1.put("position", "TL");
        soldier1.put("soldier", "../SoldierFactoryTest/TestSoldier1.json");
        soldierList.add(soldier1);
        ObjectNode soldier2 = om.createObjectNode();
        soldier2.put("position", "R");
        soldier2.put("soldier", "../SoldierFactoryTest/TestSoldier1.json");
        soldierList.add(soldier2);
        ObjectNode soldier3 = om.createObjectNode();
        soldier3.put("position", "G");
        soldier3.put("soldier", "../SoldierFactoryTest/TestSoldier1.json");
        soldierList.add(soldier3);
        ObjectNode soldier4 = om.createObjectNode();
        soldier4.put("position", "AR");
        soldier4.put("soldier", "../SoldierFactoryTest/TestSoldier1.json");
        soldierList.add(soldier4);
```

```
        ObjectNode fireteam = om.createObjectNode();
        fireteam.put("class", "mil.sstaf.core.entity.Unit");
        fireteam.put("name", "Test FT");
        fireteam.put("type", "FireTeam");
        Path fakePath = Path.of(System.getProperty("user.dir"), "src/test/resources/UnitFactoryTe
st");
        fireteam.set("soldiers", soldierList);
        fireteam.set("features", om.createArrayNode());
        fireteam.set("configurations", om.createObjectNode());
        Path p  = Path.of(RESOURCE_DIR.toString(),"fake.json");
        Unit unit = Unit.from(fireteam, p);
        assertNotNull(unit);
        assertEquals(4, unit.getAllMembers().size());
    }
    @Test
    void badJSONThrows() {
        ObjectMapper om = new ObjectMapper();
        assertThrows(SSTAFException.class, () -> {
            ObjectNode objectNode = om.createObjectNode();
            Unit unit = Unit.from(objectNode, RESOURCE_DIR
            );
            assertNotNull(unit);
        });
    }
    @Test
    void missingFileThrows() {
        assertThrows(SSTAFException.class, () -> {
                    String filename = "NotThere.json";
                    Unit unit = Unit.from(makeFile(filename));
                    assertNull(unit);
                }
        );
    }
    @Test
    void badFileThrows() {
        assertThrows(SSTAFException.class, () -> {
                    String filename = "BadUnit1.json";
                    Unit unit = Unit.from(makeFile(filename));
                    assertNull(unit);
                }
        );
    }
    @Test
    void soldierWithoutPositionThrows() {
        assertThrows(SSTAFException.class, () -> {
                    String filename = "BadUnit2.json";
                    Unit unit = Unit.from(makeFile(filename));
                    assertNull(unit);
                }
        );
    }
    @Test
    void soldierWithoutDefinitionThrows() {
        assertThrows(SSTAFException.class, () -> {
                    String filename = "BadUnit3.json";
                    Unit unit = Unit.from(makeFile(filename));
                    assertNull(unit);
                }
        );
    }
    @Test
    void unitWithoutLabelThrows() {
        assertThrows(SSTAFException.class, () -> {
                    String filename = "BadUnit4.json";
                    Unit unit = Unit.from(makeFile(filename));
                    assertNull(unit);
                }
        );
```

```
    }
    @Test
    void unitWithoutDefinitionThrows() {
        assertThrows(SSTAFException.class, () -> {
                    String filename = "BadUnit5.json";
                    Unit unit = Unit.from(makeFile(filename));
                    assertNull(unit);
                }
        );
    }
}
```

A.298 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/entity/UnitTest.java

```
package mil.sstaf.core.entity;
import org.junit.jupiter.api.Test;
import java.util.Map;
import static org.junit.jupiter.api.Assertions.*;
class UnitTest {
    @Test
    void canCreateDefaultUnit() {
        Unit unit = Unit.builder().build();
        assertNotNull(unit);
        assertTrue(unit.getName()
                .matches("([a-fA-F0-9]{8}-[a-fA-F0-9]{4}-[a-fA-F0-9]{4}-[a-fA-F0-9]{4}-[a-fA-F0-9
]{12}){1}"));
        unit.setName("It");
        assertEquals("It", unit.getName());
    }
    private Soldier makeFreePeople(final String def, final String name) {
        var d1b = Soldier.builder();
        d1b.definitionName(def);
        d1b.name(name);
        return d1b.build();
    }
    private Soldier makeDwarf(final String name) {
        return makeFreePeople("Dwarf", name);
    }
    private Soldier makeHobbit(final String name) {
        return makeFreePeople("Hobbit", name);
    }
    @Test
    void canBuildAndManipulateUnitHierarchy() {

        var d1 = makeDwarf("Thorin");
        var d2 = makeDwarf("Balin");
        var d3 = makeDwarf("Dwalin");
        var d4 = makeDwarf("Oin");
        var d5 = makeDwarf("Gloin");
        var d6 = makeDwarf("Kili");
        var d7 = makeDwarf("Fili");
        var d8 = makeDwarf("Bifur");
        var d9 = makeDwarf("Bofur");
        var d10 = makeDwarf("Bombur");
        var d11 = makeDwarf("Ori");
        var d12 = makeDwarf("Dori");
        var d13 = makeDwarf("Nori");
        var h1 = makeDwarf("Hobbit");
        var ft1b = Unit.builder();
        ft1b.type(UnitType.FireTeam);
        ft1b.soldier(Unit.MemberSoldier.of("Commander", d10));
        ft1b.soldier(Unit.MemberSoldier.of("Ax 1", d11));
        ft1b.soldier(Unit.MemberSoldier.of("Ax 2", d12));
        ft1b.soldier(Unit.MemberSoldier.of("Miner", d13));
        ft1b.name("Fire Team 1");
        var ft1 = ft1b.build();
        var ft2b = Unit.builder();
        ft2b.type(UnitType.FireTeam);
        ft2b.name("Fire Team 2");
        ft2b.soldier(Unit.MemberSoldier.of("Commander", d6));
        ft2b.soldier(Unit.MemberSoldier.of("Ax 1", d7));
        ft2b.soldier(Unit.MemberSoldier.of("Ax 2", d8));
        ft2b.soldier(Unit.MemberSoldier.of("Miner", d9));
        var ft2 = ft2b.build();
        var ft3b = Unit.builder();
        ft3b.type(UnitType.FireTeam);
```

```
        ft3b.name("Fire Team 3");
        ft3b.soldier(Unit.MemberSoldier.of("Commander", d2));
        ft3b.soldier(Unit.MemberSoldier.of("Ax 1", d3));
        ft3b.soldier(Unit.MemberSoldier.of("Ax 2", d4));
        ft3b.soldier(Unit.MemberSoldier.of("Miner", d5));
        var ft3 = ft3b.build();
        var sq1b = Unit.builder();
        sq1b.type(UnitType.Squad);
        sq1b.name("Thorin's Squad");
        sq1b.subUnit(Unit.MemberUnit.of("Fire Team 1", ft1));
        sq1b.soldier(Unit.MemberSoldier.of("G2", h1));
        sq1b.soldier(Unit.MemberSoldier.of("Commander", d1));
        var sq1 = sq1b.build();
        assertEquals(6, sq1.getNumMembers());
        sq1.addUnit(ft2.getName(), ft2);
        sq1.addUnit(ft3.getName(), ft3);
        assertEquals(UnitType.Squad, sq1.getType());
        assertEquals(UnitType.FireTeam, ft1.getType());
        assertEquals(UnitType.FireTeam, ft2.getType());
        assertEquals(UnitType.FireTeam, ft3.getType());

        //
        // Check count
        //
        assertEquals(14, sq1.getNumMembers());
        assertEquals(14, sq1.getAllMembers().size());
        assertNotNull(sq1.getSoldierByName("Kili"));
        assertEquals(d10, sq1.getSoldierByName("Bombur"));
        assertNull(sq1.getSoldierByName("Gandalf"));
        assertNotNull(ft1.getSoldierByName("Ori"));
        ft1.transferSoldierTo("Ori", ft3, "Cook");
        assertNotNull(ft3.getSoldierByName("Ori"));
        assertNull(ft1.getSoldierByName("Ori"));
        assertNotNull(sq1.getSoldierByName("Ori"));
        Unit where = sq1.getSoldierByName("Ori").getUnit();
        assertNotNull(where);
        where.removeSoldierByName("Ori");
        assertNull(sq1.getSoldierByName("Ori"));
        assertEquals(13, sq1.getNumMembers());
        sq1.setCommanderByName(h1.getName());
        assertEquals(h1, sq1.getCommander());
        sq1.setCommanderByName("Ori");
        assertEquals(h1, sq1.getCommander());
        var h2 = makeHobbit("Frodo");
        var h3 = makeHobbit("Sam");
        var h4 = makeHobbit("Merry");
        var h5 = makeHobbit("Pippin");
        var ft4b = Unit.builder();
        ft4b.type(UnitType.FireTeam);
        ft4b.name("Ring Team");
        ft4b.soldier(Unit.MemberSoldier.of("Commander", h2));
        ft4b.soldier(Unit.MemberSoldier.of("Gardener", h3));
        ft4b.soldier(Unit.MemberSoldier.of("Wraith Stabber", h4));
        ft4b.soldier(Unit.MemberSoldier.of("Kid", h5));
        var ft4 = ft4b.build();
        sq1.addUnit(ft4.getName(), ft4);
        assertEquals("Gardener", sq1.getPositionForSoldierByName("Sam"));
        assertEquals(17, sq1.getNumMembers());
        assertEquals(h3, sq1.getSoldierByName("Sam"));
        Soldier found1 = sq1.getSoldierByPosition("Thorin's Squad:Ring Team:Kid");
        assertEquals(h5, found1);
        Soldier notFound = sq1.getSoldierByPosition("Thorin's Squad:Ring Team:Ax 1");
        assertNull(notFound);
        notFound = sq1.getSoldierByPosition("::Ring Team:Ax 1");
        assertNull(notFound);
        Soldier found2 = sq1.getSoldierByPosition("Thorin's Squad:Commander");
        assertEquals(h1, found2);
        Soldier found3 = sq1.getSoldierByPosition("Commander");
```

```
        assertEquals(h1, found3);
        notFound = sq1.getSoldierByPosition("Thorin's Squad:Wraiths:Witch King");
        assertNull(notFound);
        Map<String, Unit> all = sq1.getAllUnits();
        assertNotNull(all);
        assertEquals(5, all.size());
        assertTrue(all.containsKey("Thorin's Squad:Ring Team"));
        int countBefore = ft1.getNumMembers();
        ft1.transferSoldierTo("Rosie", ft2, "Bar Maid");
        assertEquals(countBefore, ft1.getNumMembers());
        Soldier h6 = makeHobbit("Fatty");
        assertNull(sq1.getPositionForSoldierByName("Tom Cotton"));
        ft4.addSoldier("Decoy", h6);
        assertEquals(5, ft4.getNumMembers());
        ft4.removeSoldierByName("Frodo");
        assertEquals(4, ft4.getNumMembers());
        ft4.setCommanderByName("Fatty");
        assertEquals(h6, ft4.getCommander());
        h6.setUnit(ft4, null);
        assertThrows(IllegalStateException.class, () -> ft4.removeSoldierByName("Fatty"));
        h6.setUnit(null, "Decoy");
        assertThrows(IllegalStateException.class, () -> ft4.removeSoldierByName("Fatty"));
        ft4.detach();
        assertEquals(13, sq1.getNumMembers());
    }
    @Test
    void checkThatCheckerWork() {
        var h2 = makeHobbit("Frodo");
        var h3 = makeHobbit("Sam");
        var h4 = makeHobbit("Merry");
        var h5 = makeHobbit("Pippin");
        var ft4b = Unit.builder();
        ft4b.type(UnitType.FireTeam);
        ft4b.name("Ring Team");
        ft4b.soldier(Unit.MemberSoldier.of("Commander", h2));
        ft4b.soldier(Unit.MemberSoldier.of("Gardener", h3));
        ft4b.soldier(Unit.MemberSoldier.of("Wraith Stabber", h4));
        ft4b.soldier(Unit.MemberSoldier.of("Kid", h5));
        var ft4 = ft4b.build();
        assertEquals(4, ft4.getNumMembers());
        assertThrows(NullPointerException.class, () -> ft4.removeSoldierByName(null));
        assertThrows(NullPointerException.class, () -> ft4.getSoldierByName(null));
        assertThrows(NullPointerException.class, () -> ft4.getSoldierByPosition(null));
        assertThrows(NullPointerException.class, () -> ft4.setCommanderByName(null));
        assertThrows(NullPointerException.class, () -> ft4.addSoldier(null, h5));
        assertThrows(NullPointerException.class, () -> ft4.addSoldier(null));
        assertThrows(NullPointerException.class, () -> ft4.addSoldier("Boss", null));
        assertThrows(NullPointerException.class, () -> ft4.addUnit(null, null));
    }
    @Test
    void testBuilderBehavior() {
        var builder = Unit.builder();
        assertThrows(NullPointerException.class, () -> {
            Soldier s = Soldier.builder().build();
            builder.soldier(Unit.MemberSoldier.of(null, s));
        });
        assertThrows(NullPointerException.class, () -> builder.soldier(Unit.MemberSoldier.of("Com
mander", null)));
        assertThrows(NullPointerException.class, () -> {
            Unit u = Unit.builder().build();
            builder.subUnit(Unit.MemberUnit.of(null, u));
        });
        assertDoesNotThrow(() -> {
            Unit u = Unit.builder().build();
            builder.subUnit(Unit.MemberUnit.of("Fred", u));
        });
        assertDoesNotThrow(() -> {
            var b = Unit.builder();
```

```
            b.name("a");
            Unit u = b.build();
            builder.subUnit(Unit.MemberUnit.of("Unit", u));
        });
        assertThrows(NullPointerException.class, () -> {
            Unit.builder().build();
            builder.subUnit(Unit.MemberUnit.of("Fire Team", null));
        });
    }
}
```

A.299 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/features/ConfigurationTest.java

```
package mil.sstaf.core.features;
import com.fasterxml.jackson.databind.ObjectMapper;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Nested;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertDoesNotThrow;
import static org.junit.jupiter.api.Assertions.assertEquals;
public class ConfigurationTest {
    @Nested
    @DisplayName("Test the happy path")
    class HappyTests {
        @Test
        @DisplayName("Confirm that an empty configuration works and results in seed == 0")
        public void test1() {
            String jsonString = "{ \"class\" : \"" + FeatureConfiguration.class.getName() + "\"}"
;
            ObjectMapper mapper = new ObjectMapper();
            assertDoesNotThrow(() -> {
                FeatureConfiguration configuration = mapper.readValue(jsonString, FeatureConfigur
ation.class);
                assertEquals(0, configuration.getSeed());
            });
        }
        @Test
        @DisplayName("Confirm that setting the seed in the JSON works")
        public void test2() {
            String jsonString = "{ \"class\" : \"" + FeatureConfiguration.class.getName() + "\" ,
\"seed\" : 4569870123 }";
            ObjectMapper mapper = new ObjectMapper();
            assertDoesNotThrow(() -> {
                FeatureConfiguration configuration = mapper.readValue(jsonString, FeatureConfigur
ation.class);
                assertEquals(4569870123L, configuration.getSeed());
            });
        }

    }
}
```

A.300 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/features/FeatureLoaderTest.java

```
package mil.sstaf.core.features;

import mil.sstaf.core.entity.Address;
import mil.sstaf.core.configuration.SSTAFConfiguration;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import java.util.List;
import java.util.Optional;

import static org.junit.jupiter.api.Assertions.*;

class FeatureLoaderTest {

    @BeforeEach
    void setUp() {
        FeatureLoader.registerClass(BananaFeature.class);
        FeatureLoader.registerClass(HarryHandler.class);
        FeatureLoader.registerClass(AliceAgent.class);
    }

    @AfterEach
    void reset() {
        FeatureLoader.clearClassRegistry();
    }

    @Test
    void canRegisterAProvider() {
        assertEquals(3, FeatureLoader.getRegisteredClasses().size());
    }

    @Test
    void canInstantiateClasses() {
        FeatureLoader<BananaFeature> featureLoader1 = new FeatureLoader<>(BananaFeature.class, th
is, ModuleLayer.boot());
        Assertions.assertNotNull(featureLoader1);

        Assertions.assertDoesNotThrow(() -> {
            List<BananaFeature> candidates = featureLoader1.generateFeatureInstances();
            assertEquals(1, candidates.size());
            BananaFeature bananaFeature = candidates.get(0);
            assertEquals(1, bananaFeature.getMajorVersion());
            assertEquals(2, bananaFeature.getMinorVersion());
            assertEquals(3, bananaFeature.getPatchVersion());
        });

        FeatureLoader<HarryHandler> featureLoader2 = new FeatureLoader<>(HarryHandler.class, this
, ModuleLayer.boot());
        Assertions.assertNotNull(featureLoader2);

        Assertions.assertDoesNotThrow(() -> {
            List<HarryHandler> candidates = featureLoader2.generateFeatureInstances();
            assertEquals(1, candidates.size());
            HarryHandler harryHandler = candidates.get(0);
            assertEquals(5, harryHandler.getMajorVersion());
            assertEquals(3, harryHandler.getMinorVersion());
            assertEquals(1, harryHandler.getPatchVersion());
        });
```

```
        FeatureLoader<AliceAgent> featureLoader3 = new FeatureLoader<>(AliceAgent.class, this, Mo
duleLayer.boot());
        Assertions.assertNotNull(featureLoader3);

        Assertions.assertDoesNotThrow(() -> {
            List<AliceAgent> candidates = featureLoader3.generateFeatureInstances();
            assertEquals(1, candidates.size());
            AliceAgent AliceAgent = candidates.get(0);
            assertEquals(1, AliceAgent.getMajorVersion());
            assertEquals(1, AliceAgent.getMinorVersion());
            assertEquals(1, AliceAgent.getPatchVersion());
        });

        FeatureLoader<Feature> featureLoader4 = new FeatureLoader<>(Feature.class, this, ModuleLa
yer.boot());
        Assertions.assertNotNull(featureLoader4);

        Assertions.assertDoesNotThrow(() -> {
            List<Feature> candidates = featureLoader4.generateFeatureInstances();
            assertEquals(3, candidates.size());
        });
    }

    @Test
    void meetsRequirementsWorks() {
        HarryHandler hh1 = new HarryHandler();
        HarryHandler hh2 = new HarryHandler2();
        HarryHandler hh3 = new HarryHandler3();

        assertFalse(FeatureLoader.meetsRequirements(null, "Harry", 4, 0, true));

        assertTrue(FeatureLoader.meetsRequirements(hh1, "Harry", 4, 0, false));
        assertFalse(FeatureLoader.meetsRequirements(hh1, "Harry", 4, 0, true));
        assertTrue(FeatureLoader.meetsRequirements(hh1, "Harry", 5, 3, true));

        assertTrue(FeatureLoader.meetsRequirements(hh2, "Harry", 4, 0, false));
        assertFalse(FeatureLoader.meetsRequirements(hh2, "Harry", 5, 3, true));
        assertTrue(FeatureLoader.meetsRequirements(hh2, "Harry", 5, 4, true));

        assertTrue(FeatureLoader.meetsRequirements(hh3, "Harry", 4, 0, false));
        assertFalse(FeatureLoader.meetsRequirements(hh3, "Harry", 4, 0, true));
        assertTrue(FeatureLoader.meetsRequirements(hh3, "Harry", 6, 0, true));


    }

    @Test
    void compareVersionsWorks() {
        HarryHandler harryHandler = new HarryHandler();
        HarryHandler harryHandler2 = new HarryHandler2();
        HarryHandler harryHandler3 = new HarryHandler3();

        assertEquals(0, FeatureLoader.compareVersions(harryHandler, harryHandler));
        assertEquals(0, FeatureLoader.compareVersions(harryHandler2, harryHandler2));
        assertEquals(0, FeatureLoader.compareVersions(harryHandler3, harryHandler3));

        assertEquals(-1, FeatureLoader.compareVersions(harryHandler2, harryHandler));
        assertEquals(-1, FeatureLoader.compareVersions(harryHandler3, harryHandler2));
        assertEquals(-1, FeatureLoader.compareVersions(harryHandler3, harryHandler));

        assertEquals(1, FeatureLoader.compareVersions(harryHandler, harryHandler2));
        assertEquals(1, FeatureLoader.compareVersions(harryHandler2, harryHandler3));
        assertEquals(1, FeatureLoader.compareVersions(harryHandler, harryHandler3));
    }

    @Test
    void findBestMatchWorks() {
        List<HarryHandler> candidates = List.of(new HarryHandler(),
```

```
                new HarryHandler2(), new HarryHandler3());

        HarryHandler hh = FeatureLoader.findBestMatch(candidates, null, "Harry", 4, 0, false);
        assertNotNull(hh);
        assertEquals(6, hh.getMajorVersion());
        assertEquals(0, hh.getMinorVersion());
        assertEquals(0, hh.getPatchVersion());

        HarryHandler hh2 = FeatureLoader.findBestMatch(candidates, null, "Harry", 5, 4, true);
        assertNotNull(hh2);
        assertEquals(HarryHandler2.class, hh2.getClass());

    }

    @Test
    void loadWorks() {
        FeatureLoader.registerClass(HarryHandler2.class);
        FeatureLoader.registerClass(HarryHandler3.class);

        assertEquals(5, FeatureLoader.getRegisteredClasses().size());

        FeatureLoader<HarryHandler> featureLoader = new FeatureLoader<>(HarryHandler.class, this,
SSTAFConfiguration.getInstance().getRootLayer());

        Optional<HarryHandler> optionalHarryHandler =
                featureLoader.load("Harry", 4, 0, false);
        assertTrue(optionalHarryHandler.isPresent());
        HarryHandler hh = optionalHarryHandler.get();
        assertEquals(HarryHandler3.class, hh.getClass());
    }


    static class BananaFeature extends BaseFeature {

        public BananaFeature() {
            super("Banana", 1, 2, 3, false, "Flingin' Poo!");
        }

    }

    static class HarryHandler extends BaseHandler {

        public HarryHandler() {
            super("Harry", 5, 3, 1, false, "I handle integers with panache");
        }

        protected HarryHandler(String featureName, int majorVersion, int minorVersion, int patchV
ersion, boolean requiresConfiguration, String description) {
            super(featureName, majorVersion, minorVersion, patchVersion, requiresConfiguration, d
escription);
        }

        @Override
        public List<Class<? extends HandlerContent>> contentHandled() {
            return List.of(IntContent.class);
        }

        @Override
        public ProcessingResult process(HandlerContent arg, long scheduledTime_ms, long currentTi
me_ms, Address from, long id, Address respondTo) {
            return null;
        }

    }

    static class HarryHandler2 extends HarryHandler {

        public HarryHandler2() {
```

```
            super("Harry", 5, 4, 1, false, "");
        }

    }

    static class HarryHandler3 extends HarryHandler {

        public HarryHandler3() {
            super("Harry", 6, 0, 0, false, "");
        }
    }

    static class AliceAgent extends BaseAgent {

        public AliceAgent() {
            super("Alice", 1, 1, 1, false,
                    "You can get anything you want.");
        }

        @Override
        public ProcessingResult tick(long currentTime_ms) {
            return ProcessingResult.empty();
        }

        @Override
        public List<Class<? extends HandlerContent>> contentHandled() {
            return List.of();
        }

        @Override
        public ProcessingResult process(HandlerContent arg, long scheduledTime_ms, long currentTi
me_ms, Address from, long id, Address respondTo) {
            return null;
        }

    }
}
```

A.301 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/features/FeatureSpecificationFactoryTest.java

```
package mil.sstaf.core.features;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.node.ObjectNode;
import mil.sstaf.core.util.SSTAFException;
import org.junit.jupiter.api.Test;
import java.io.File;
import java.nio.file.Path;
import static org.junit.jupiter.api.Assertions.*;
class FeatureSpecificationFactoryTest {

    static Path USER_DIR = Path.of(System.getProperty("user.dir"));
    static Path RESOURCE_DIR = Path.of(USER_DIR.toString(), "src/test/resources");
    @Test
    void readingFromGoodFileWorks() {
        String file = "src/test/resources/TestProviderSpec.json";
        FeatureSpecification ps = FeatureSpecification.from(new File(file));
        assertEquals("Mary Poppins", ps.featureName);
        assertEquals(3, ps.majorVersion);
        assertEquals(1, ps.minorVersion);
        assertTrue(ps.requireExact);
    }
    @Test
    void readingFromBadFileThrows() {
        assertThrows(SSTAFException.class, () -> {
            String file = "src/test/resources/NotATestProviderSpec.json";
            FeatureSpecification.from(new File(file));
        });
    }
    @Test
    void badArgumentThrows() {
        assertThrows(SSTAFException.class, () -> {
            ObjectMapper om = new ObjectMapper();
            FeatureSpecification.from(om.createArrayNode(), RESOURCE_DIR);
        });
    }
    public static final String NAME_KEY = "featureName";
    public static final String MAJOR_VERSION_KEY = "majorVersion";
    public static final String MINOR_VERSION_KEY = "minorVersion";
    public static final String REQUIRE_EXACT_KEY = "requireExact";
    @Test
    void badJSONThrows1() {
        ObjectMapper om = new ObjectMapper();
        assertThrows(SSTAFException.class, () -> {
            ObjectNode testArticle = om.createObjectNode();
            testArticle.put(MAJOR_VERSION_KEY, 3);
            testArticle.put(MINOR_VERSION_KEY, 1);
            testArticle.put(REQUIRE_EXACT_KEY, true);
            FeatureSpecification.from(testArticle, RESOURCE_DIR);
        });
    }
    @Test
    void badJSONThrows2() {
        ObjectMapper om = new ObjectMapper();
        assertThrows(SSTAFException.class, () -> {
            ObjectNode testArticle = om.createObjectNode();
            testArticle.put(NAME_KEY, "MyProvider");
            testArticle.put(MAJOR_VERSION_KEY, "Banana");
            testArticle.put(MINOR_VERSION_KEY, 1);
            testArticle.put(REQUIRE_EXACT_KEY, true);
            FeatureSpecification.from(testArticle, RESOURCE_DIR);
        });
    }
```

```
}
```

A.302 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/features/FeatureSpecificationTest.java

```
package mil.sstaf.core.features;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;
public class FeatureSpecificationTest {
    @Test
    void builderWorks() {
        var builder = FeatureSpecification.builder();
        builder.majorVersion(4)
                .minorVersion(2)
                .featureName("Wawa")
                .requireExact(true);
        FeatureSpecification ps1 = builder.build();
        Assertions.assertEquals("Wawa", ps1.featureName);
        Assertions.assertEquals(4, ps1.majorVersion);
        Assertions.assertEquals(2, ps1.minorVersion);
        Assertions.assertTrue(ps1.requireExact);
    }
}
```

A.303 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/features/ResolverTest.java

```
package mil.sstaf.core.features;

import lombok.Getter;
import lombok.experimental.SuperBuilder;
import mil.sstaf.core.entity.Address;
import mil.sstaf.core.entity.Entity;
import mil.sstaf.core.entity.EntityHandle;
import mil.sstaf.core.entity.TestEntity;
import mil.sstaf.core.util.SSTAFException;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;

import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Objects;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.ConcurrentMap;

class ResolverTest {

    @Test
    void multilayerLoadWithCircularRequirementWorks() {

        FeatureLoader.registerClass(JamesBond.class);
        FeatureLoader.registerClass(AlphaProvider.class);
        FeatureLoader.registerClass(BravoProvider.class);
        FeatureLoader.registerClass(CharlieProvider.class);
        FeatureLoader.registerClass(DeltaProvider.class);
        FeatureLoader.registerClass(EchoProvider.class);

        FeatureSpecification jbSpec = FeatureSpecification.builder()
                .featureName("James Bond")
                .majorVersion(7)
                .minorVersion(0)
                .requireExact(false).build();

        ConcurrentMap<FeatureSpecification, Feature> featureCache = new ConcurrentHashMap<>();

        Map<String, FeatureConfiguration> config = new HashMap<>();

        for (String name : new String[]{"James Bond", "Alpha", "Bravo",
                "Charlie"}) {
            config.put(name, FeatureConfiguration.builder().build());
        }

        for (String name : new String[]{"Delta", "Echo"}) {
            config.put(name, ValueConfiguration.builder().myValue(17).build());
        }
        Entity testEntity = TestEntity.makeTestEntity();

        Resolver resolver = new Resolver(featureCache, config, testEntity.getHandle(), 31415, Mod
uleLayer.boot());

        Agent jamesBond = (Agent) resolver.loadAndResolveDependencies(jbSpec);

        Assertions.assertDoesNotThrow(jamesBond::init);
    }

    static class JamesBond extends BaseAgent {

        private FeatureConfiguration configuration;
```

```
        //@SuppressWarnings()
        @Requires(name = "Alpha", majorVersion = 5, minorVersion = 3, requireExact = true)
        private AlphaProvider alphaProvider;

        @Requires(name = "Bravo", majorVersion = 2, minorVersion = 1)
        private BravoProvider bravoProvider;

        public JamesBond() {
            super("James Bond", 7, 0, 0, true,
                    "Licensed to throw exceptions");
            ownerHandle = EntityHandle.makeDummyHandle();
        }

        @Override
        public ProcessingResult tick(long currentTime_ms) {
            return null;
        }

        @Override
        public List<Class<? extends HandlerContent>> contentHandled() {
            return List.of();
        }

        @Override
        public void init() {
            super.init();
            if (configuration == null) {
                throw new SSTAFException("Null configuration");
            }
            Objects.requireNonNull(alphaProvider, "No Alpha");
            Objects.requireNonNull(bravoProvider, "No Bravo");
            alphaProvider.init();
            bravoProvider.init();
        }

        @Override
        public ProcessingResult process(HandlerContent arg, long scheduledTime_ms, long currentTi
me_ms, Address from,
                                        long id, Address respondTo) {
            return null;
        }

        @Override
        public void configure(FeatureConfiguration configuration) {
            super.configure(configuration);
            this.configuration = configuration;
        }
    }

    static class AlphaProvider extends BaseFeature {

        @SuppressWarnings("unused")
        private FeatureConfiguration configuration;

        public AlphaProvider() {
            super("Alpha", 5, 3, 0, true, "");

        }

        @Override
        public void configure(FeatureConfiguration configuration) {
            super.configure(configuration);
            this.configuration = configuration;
        }

        @Override
        public void init() throws SSTAFException {
```

```
            super.init();
            if (configuration == null) {
                throw new SSTAFException("Null configuration");
            }
        }

    }

    static class BravoProvider extends BaseFeature {

        @Requires(name = "Charlie", majorVersion = 1, minorVersion = 2, requireExact = true)
        private CharlieProvider charlieProvider;

        @Requires(name = "Delta", majorVersion = 8)
        private DeltaProvider deltaProvider;

        private FeatureConfiguration configuration;

        public BravoProvider() {
            super("Bravo", 3, 4, 0, true, "");
        }

        @Override
        public void init() throws SSTAFException {
            super.init();
            Objects.requireNonNull(charlieProvider, "No Charlie");
            Objects.requireNonNull(deltaProvider, "No Delta");
            Objects.requireNonNull(configuration, "No MapConfiguration");
            charlieProvider.init();
            deltaProvider.init();
        }

        @Override
        public void configure(FeatureConfiguration configuration) {
            super.configure(configuration);
            this.configuration = configuration;
        }
    }

    static class CharlieProvider extends BaseFeature {

        private FeatureConfiguration configuration;

        @Requires
        private EchoProvider echo;

        public CharlieProvider() {
            super("Charlie", 1, 2, 0, true, "");
        }

        @Override
        public void init() throws SSTAFException {
            super.init();
            if (configuration == null) {
                throw new SSTAFException("Null configuration");
            }
        }

        @Override
        public void configure(FeatureConfiguration configuration) {
            super.configure(configuration);
            this.configuration = configuration;
        }
    }

    @SuperBuilder
    static class ValueConfiguration extends FeatureConfiguration {
        @Getter
```

```
        int myValue;
    }

    static class DeltaProvider extends BaseFeature {

        ValueConfiguration configuration;
        int value = 4;

        public DeltaProvider() {
            super("Delta", 8, 9, 0, true, "");
        }

        @Override
        public void init() throws SSTAFException {
            super.init();
            if (configuration == null) {
                throw new SSTAFException("Null configuration");
            }
            if (value != 17) {
                throw new SSTAFException("MapConfiguration didn't work");
            }
        }

        @Override
        public void configure(FeatureConfiguration configuration) {
            super.configure(configuration);
            if (configuration instanceof ValueConfiguration) {
                this.configuration = (ValueConfiguration) configuration;
                this.value = this.configuration.getMyValue();
            }
        }
    }

    static class EchoProvider extends BaseFeature {

        ValueConfiguration configuration;
        int value = 4;

        @Requires
        BravoProvider bravo;

        public EchoProvider() {
            super("Echo", 1, 0, 0, true, "");
        }

        @Override
        public void init() throws SSTAFException {
            super.init();
            if (configuration == null) {
                throw new SSTAFException("Null configuration");
            }
            if (value != 17) {
                throw new SSTAFException("MapConfiguration didn't work");
            }
        }

        @Override
        public void configure(FeatureConfiguration configuration) {
            super.configure(configuration);
            if (configuration instanceof ValueConfiguration) {
                this.configuration = (ValueConfiguration) configuration;
                this.value = this.configuration.getMyValue();
            }
        }
    }

}
```

A.304 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/features/ResourceManagerTest.java

```
package mil.sstaf.core.features;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Nested;
import org.junit.jupiter.api.Test;
import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import static org.junit.jupiter.api.Assertions.assertNotNull;
import static org.junit.jupiter.api.Assertions.assertTrue;
public class ResourceManagerTest {
    @Nested
    @DisplayName("ResourceManager tests that don't require the assembled jar")
    class UnitTests {
        @Test
        @DisplayName(value = "Confirm that a resource at the top level can be extracted")
        public void extractionWork1() {
            try {
                Path tempDir = Files.createTempDirectory("Banana");
                File resourceFile = ResourceManager.extractResource(this.getClass(), "topLevelRes
ource.txt", tempDir);
                assertNotNull(resourceFile);
                assertTrue(resourceFile.exists());
            } catch (IOException e) {
                e.printStackTrace();
                Assertions.fail("IO!");
            }
        }
        @Test
        @DisplayName(value = "Confirm that a resource in a subdirectory can be extracted")
        public void extractionWorks2() {
            try {
                Path tempDir = Files.createTempDirectory("Banana");
                File resourceFile = ResourceManager.extractResource(this.getClass(), "extractionT
est/extractMe.txt", tempDir);
                assertNotNull(resourceFile);
                assertTrue(resourceFile.exists());
            } catch (IOException e) {
                e.printStackTrace();
                Assertions.fail("IO!");
            }
        }
        @Test
        @DisplayName(value = "Confirm that attempting to extract a non-existent resource throws I
OException")
        public void badResourceThrows() {
            Assertions.assertThrows(IOException.class, () -> {
                Path tempDir = Files.createTempDirectory("Banana");
                ResourceManager.extractResource(this.getClass(), "extractionTest/nonExistent.txt"
, tempDir);
            });
        }
        @Test
        @DisplayName(value = "Confirm that providing a null resource owner throws NullPointerExce
ption")
        public void nullResourceOwnerThrows() {
            Assertions.assertThrows(NullPointerException.class, () -> {
                Path tempDir = Files.createTempDirectory("Banana");
                ResourceManager.extractResource(null, "extractionTest/extractMe.txt", tempDir);
            });
        }
```

```
        @Test
        @DisplayName(value = "Confirm that providing a null resource name throws NullPointerExcep
tion")
        public void nullResourceNameThrows() {
            Assertions.assertThrows(NullPointerException.class, () -> {
                Path tempDir = Files.createTempDirectory("Banana");
                ResourceManager.extractResource(this.getClass(), null, tempDir);
            });
        }
        @Test
        @DisplayName(value = "Confirm that providing a null directory throws NullPointerException
")
        public void nullDirectoryThrows() {
            Assertions.assertThrows(NullPointerException.class,
                    () -> ResourceManager.extractResource(this.getClass(),
                            "extractionTest/extractMe.txt", null));
        }

    }
}
```

A.305 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/json/BaseNodeWrapperTest.java

```
package mil.sstaf.core.json;
import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.node.JsonNodeFactory;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Nested;
import org.junit.jupiter.api.Test;
import java.util.HashMap;
import java.util.Map;
import static org.junit.jupiter.api.Assertions.*;
public class BaseNodeWrapperTest {
    ObjectMapperFactory omf = path -> null;
    Map<String, JsonNode> referenceCache = new HashMap<>();
    static class FakeWrapper extends BaseNodeWrapper<JsonNode> {
        protected FakeWrapper(NodeWrapper parent, JsonNode node, ObjectMapperFactory objectMapper
Factory, Map<String, JsonNode> referenceCache) {
            super(parent, node, objectMapperFactory, referenceCache);
        }
        @Override
        public void resolve() {
        }
        @Override
        public void replaceNode(JsonNode original, JsonNode replacement) {
        }
    }
    @Nested
    @DisplayName("Test the 'Happy Path'")
    class HappyTests {
        @Test
        @DisplayName("Confirm that refCount works for non-reference Wrappers")
        public void refCountTest1() {
            FakeWrapper one = new FakeWrapper(null, null, omf, referenceCache);
            FakeWrapper two = new FakeWrapper(one, null, omf, referenceCache);
            FakeWrapper three = new FakeWrapper(two, null, omf, referenceCache);
            FakeWrapper four = new FakeWrapper(three, null, omf, referenceCache);
            assertEquals(0, four.refCount("bob"));
            assertEquals(0, four.refCount("diane"));
            assertEquals(0, four.refCount("luigi"));
        }
        @Test
        @DisplayName("Confirm that isResolved() works as expected.")
        public void isResolvedTest1() {
            FakeWrapper fakeWrapper = new FakeWrapper(null, null, omf, referenceCache);
            assertFalse(fakeWrapper.isResolved());
            fakeWrapper.resolved = true;
            assertTrue(fakeWrapper.isResolved());
            fakeWrapper.resolved = false;
            assertFalse(fakeWrapper.isResolved());
        }
        @Test
        @DisplayName("Confirm that getNode() works as expected")
        public void getNodeTest1() {
            JsonNodeFactory jsonNodeFactory = new JsonNodeFactory(true);
            JsonNode textNode = jsonNodeFactory.textNode("This is the text");
            FakeWrapper fakeWrapper = new FakeWrapper(null, textNode, omf, referenceCache);
            assertEquals(textNode, fakeWrapper.getMyNode());
        }
        @Test
        @DisplayName("Confirm that getParent() works as expected")
        public void getParentTest1() {
            FakeWrapper one = new FakeWrapper(null, null, omf, referenceCache);
            FakeWrapper two = new FakeWrapper(one, null, omf, referenceCache);
            FakeWrapper three = new FakeWrapper(two, null, omf, referenceCache);
```

```
            FakeWrapper four = new FakeWrapper(three, null, omf, referenceCache);
            assertEquals(three, four.getParent());
            assertEquals(two, three.getParent());
            assertEquals(one, two.getParent());
            assertNull(one.getParent());
        }

    }
}
```

A.306 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/json/EntityHandleSerializerTest.java

```
package mil.sstaf.core.json;
import com.fasterxml.jackson.databind.ObjectMapper;
import mil.sstaf.core.entity.EntityHandle;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertTrue;
public class EntityHandleSerializerTest {
    @Test
    @DisplayName("Confirm that it works as expected")
    void test1() {
        Assertions.assertDoesNotThrow(() -> {
            EntityHandle entityHandle = EntityHandle.makeDummyHandle();
            ObjectMapper objectMapper = new ObjectMapper();
            String out = objectMapper.writeValueAsString(entityHandle);
            assertTrue(out.contains(entityHandle.getPath()));
        });
    }
}
```

A.307 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/json/JsonLoaderTest.java

```
package mil.sstaf.core.json;
import mil.sstaf.core.util.SSTAFException;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Nested;
import org.junit.jupiter.api.Test;
import java.nio.file.Path;
import java.util.List;
import static org.junit.jupiter.api.Assertions.*;
public class JsonLoaderTest {

    public static final String PATH_STRING = "src/test/resources/jsonTests";
    public static final Path DEFAULT_PATH = Path.of("src/test/resources/jsonTests");
    private Path makePath(String file) {
        return Path.of(PATH_STRING, file).normalize();
    }
    @Nested
    @DisplayName("Test the 'Happy Path'")
    class HappyTests {
        @Test
        @DisplayName("Confirm that a simple JSON string can be processed")
        void loadTest1() {
            Assertions.assertDoesNotThrow(() -> {
                String x = "{ \"class\":\"mil.sstaf.core.json.Message1\", \"strings\":[]}";
                JsonLoader loader = new JsonLoader();
                Message1 msg1 = (Message1) loader.load(x, DEFAULT_PATH);
                Assertions.assertNotNull(msg1);
            });
        }
        @Test
        @DisplayName("Confirm that an object graph that contains references can be loaded")
        void loadTest2() {
            assertDoesNotThrow(() -> {
                JsonLoader jsonLoader = new JsonLoader();
                Message6 message6 = (Message6) jsonLoader.load(makePath("message6.json"));
                assertNotNull(message6);
                assertNotNull(message6.getFours());
                List<Message4> fours = message6.getFours();
                assertEquals(4, fours.size());
                for (Message4 msg4 : fours) {
                    assertNotNull(msg4.getIt());
                    Message2 msg2 = msg4.getIt();
                    assertEquals(6, msg2.getInts().size());
                }
            });
        }
        @Test
        @DisplayName("Confirm that absolute paths work during construction")
        void loadTest3() {
            assertDoesNotThrow(() -> {
                Path fullPath = Path.of(System.getProperty("user.dir"), PATH_STRING, "message6.js
on").toAbsolutePath();
                JsonLoader jsonLoader = new JsonLoader();
                Message6 message6 = (Message6) jsonLoader.load(fullPath);
                assertNotNull(message6);
                assertNotNull(message6.getFours());
                List<Message4> fours = message6.getFours();
                assertEquals(4, fours.size());
                for (Message4 msg4 : fours) {
                    assertNotNull(msg4.getIt());
                    Message2 msg2 = msg4.getIt();
                    assertEquals(6, msg2.getInts().size());
```

```
                }
            });
        }
        @Test
        @DisplayName("Confirm that absolute paths work when loading")
        void loadTest4() {
            assertDoesNotThrow(() -> {
                Path fullPath = Path.of(System.getProperty("user.dir"), PATH_STRING, "message6.js
on");
                JsonLoader jsonLoader = new JsonLoader();
                Message6 message6 = (Message6) jsonLoader.load(fullPath);
                assertNotNull(message6);
                assertNotNull(message6.getFours());
                List<Message4> fours = message6.getFours();
                assertEquals(4, fours.size());
                for (Message4 msg4 : fours) {
                    assertNotNull(msg4.getIt());
                    Message2 msg2 = msg4.getIt();
                    assertEquals(6, msg2.getInts().size());
                }
            });
        }
    }
    @Nested
    @DisplayName("Test failure modes")
    class FailureTests {
        @Test
        @DisplayName("Confirm that circular references throw an exception.")
        void resolveTest5() {
            JsonResolutionException jre = assertThrows(JsonResolutionException.class, () -> {
                JsonLoader loader = new JsonLoader();
                Path topFile = makePath("message8.json");
                loader.load(topFile);
            });
            assertTrue(jre.getMessage().contains("Circular"));
        }
        @Test
        @DisplayName("Confirm that bad references throw an exception.")
        void resolveTest6() {
            JsonResolutionException jre = assertThrows(JsonResolutionException.class, () -> {
                JsonLoader loader = new JsonLoader();
                Path topFile = makePath("badref.json");
                loader.load(topFile);
            });
            assertTrue(jre.getMessage().contains("Could not load reference"));
            assertTrue(jre.getMessage().contains("not_there.json"));
        }
        @Test
        @DisplayName("Confirm that unknown classes throw an exception.")
        void resolveTest7() {
            SSTAFException se = assertThrows(SSTAFException.class, () -> {
                JsonLoader loader = new JsonLoader();
                Path topFile =makePath("badclass.json");
                loader.load(topFile);
            });
            assertTrue(se.getMessage().contains("Could not load"));
            assertTrue(se.getCause() instanceof ClassNotFoundException);
        }
        @Test
        @DisplayName("Confirm that a JSON object without a class field throws.")
        void resolveTest8() {
            SSTAFException se = assertThrows(SSTAFException.class, () -> {
                JsonLoader loader = new JsonLoader();
                Path topFile = makePath("noClassField.json");
                loader.load(topFile);
            });
            assertTrue(se.getMessage().contains("JSON does not specify a class name"));
        }
```

```
    }
}
```

A.308 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/json/JsonNodeContextTest.java

```
package mil.sstaf.core.json;
import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;
import java.util.HashMap;
import static org.junit.jupiter.api.Assertions.*;
public class JsonNodeContextTest {

    @Test
    public void test1() {
//
//        String json = "{ \"a\" " + " : " + "\"theFile.json\" }";
//
//        assertDoesNotThrow(() -> {
//            ObjectMapper objectMapper = new ObjectMapper();
//            JsonNode node = objectMapper.readTree(json);
//
//            JsonNodeReference ctx = new JsonNodeReference(node, null, new HashMap<String, JsonN
odeReference>());
//
//            Assertions.assertNull(ctx.getDirectory());
//            Assertions.assertEquals(node, ctx.getRootNode());
//            Assertions.assertNull(ctx.getParent());
//
//            var bob = ctx.identifyReferences();
//            Assertions.assertEquals(1, bob.size());
//            JsonNodeReference ctx2 = bob.get(0);
//            Assertions.assertEquals("theFile.json", ctx2.getReference());
//
//        });

    }
}
```

A.309 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/json/Message1.java

```
package mil.sstaf.core.json;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.Getter;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import java.util.List;
@Jacksonized
@SuperBuilder
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
public class Message1 {
    @Getter
    private final List<String> strings;
}
```

A.310 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/json/Message2.java

```
package mil.sstaf.core.json;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.Getter;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import java.util.List;
@SuperBuilder
@Jacksonized
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
public class Message2 extends Message1 {
    @Getter
    private final List<Integer> ints;
}
```

A.311 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/json/Message3.java

```
package mil.sstaf.core.json;

import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.Getter;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import java.util.List;
@Jacksonized
@SuperBuilder
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
public class Message3 {
    @Getter
    List<Message1> msgs;
}
```

A.312 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/json/Message4.java

```
package mil.sstaf.core.json;

import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.Getter;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
@Jacksonized
@SuperBuilder
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
public class Message4 {
   @Getter
    Message2 it;
}
```

A.313 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/json/Message5.java

```
package mil.sstaf.core.json;

import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.Getter;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
@Jacksonized
@SuperBuilder
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
public class Message5 {
   @Getter
    Message4 msg4;
}
```

A.314 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/json/Message6.java

```
package mil.sstaf.core.json;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.Getter;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import java.util.List;
@SuperBuilder
@Jacksonized
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
public class Message6 extends Message1 {
    @Getter
    private final List<Message4> fours;
}
```

A.315 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/json/Message7.java

```
package mil.sstaf.core.json;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.Getter;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
@SuperBuilder
@Jacksonized
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
public class Message7 extends Message1 {
    @Getter
    private final Message8 msg8;
}
```

A.316 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/json/Message8.java

```
package mil.sstaf.core.json;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.Getter;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
@SuperBuilder
@Jacksonized
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
public class Message8 extends Message1 {
    @Getter
    private final Message7 msg7;
}
```

A.317 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/json/TestSimpleThings.java

```
package mil.sstaf.core.json;
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Nested;
import org.junit.jupiter.api.Test;
import java.util.List;
import java.util.Random;
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertNotNull;
public class TestSimpleThings {
    private static final char QUOTE = '"';
    private static final char COLON = ':';
    private static final char COMMA = ',';
    private static final char START_OBJ = '{';
    private static final char END_OBJ = '}';
    private static final char START_LIST = '[';
    private static final char END_LIST = ']';
    private static final String CN = "class";
    private static final int NUM_STRINGS = 4;
    private static final int NUM_INTS = 7;
    private static final int NUM_MSGS = 12;
    final Random random = new Random(987654321L);
    String makeClassName(final Class<?> c) {
        return QUOTE + CN + QUOTE + COLON + QUOTE + c.getName() + QUOTE;
    }
    String wrapObj(String contents) {
        return START_OBJ + contents + END_OBJ;
    }
    String makeMessage1Contents() {
        String[] names = new String[]{
                "Thorin", "Balin", "Dwalin", "Kili", "Fili", "Dori", "Nori", "Ori",
                "Bifur", "Bofur", "Bombur", "Oin", "Gloin", "Bilbo", "Ganfalf"};
        StringBuilder sb = new StringBuilder();
        sb.append(QUOTE).append("strings").append(QUOTE).append(COLON);
        sb.append(START_LIST);
        for (int i = 0; i < NUM_STRINGS ; ++i) {
            String s = names[random.nextInt(names.length)];
            sb.append(QUOTE).append(s).append(QUOTE);
            if (i < (NUM_STRINGS - 1 )) sb.append(COMMA);
        }
        sb.append(END_LIST);
        return sb.toString();
    }
    String makeJSONForMessage1() {
        String sb = makeClassName(Message1.class) + COMMA +
                makeMessage1Contents();
       return wrapObj(sb);
    }
    String makeJSONForMessage2() {
        StringBuilder sb = new StringBuilder();
        sb.append(makeClassName(Message2.class)).append(COMMA);
        sb.append(makeMessage1Contents()).append(COMMA);
        sb.append(QUOTE).append("ints").append(QUOTE).append(COLON);
        sb.append(START_LIST);
        for (int i = 0; i < NUM_INTS; ++i) {
            sb.append(random.nextInt());
            if (i < NUM_INTS - 1) sb.append(COMMA);
        }
        sb.append(END_LIST);
```

```
        return wrapObj(sb.toString());
    }
    String makeJSONForMessage3() {
        StringBuilder sb = new StringBuilder();
        sb.append(makeClassName(Message3.class)).append(COMMA);
        sb.append(QUOTE).append("msgs").append(QUOTE).append(COLON);
        sb.append(START_LIST);
        for (int i = 0; i < NUM_MSGS; ++i) {
            sb.append( i %2 == 0 ? makeJSONForMessage2() : makeJSONForMessage1());
            if (i < NUM_MSGS - 1) sb.append(COMMA);
        }
        sb.append(END_LIST);
        return wrapObj(sb.toString());
    }
    @Nested
    @DisplayName("Test scenarios where the input is valid (The happy path)")
    class HappyTests {
        @Test
        @DisplayName("Confirm that deserializing Message1 works")
        public void test1() throws JsonProcessingException {
            String jsonString = makeJSONForMessage1();
            ObjectMapper objectMapper = new ObjectMapper();
            Message1 message1 = objectMapper.readValue(jsonString, Message1.class);
            assertNotNull(message1);
            assertEquals(NUM_STRINGS, message1.getStrings().size());
        }
        @Test
        @DisplayName("Confirm that deserializing Message2 works")
        public void test2() throws JsonProcessingException {
            String jsonString = makeJSONForMessage2();
            ObjectMapper objectMapper = new ObjectMapper();
            Message2 message2 = objectMapper.readValue(jsonString, Message2.class);
            assertNotNull(message2);
            assertEquals(NUM_STRINGS, message2.getStrings().size());
            assertEquals(NUM_INTS, message2.getInts().size());
        }
        private String getCurrentStackFrame() {
            Thread t = Thread.currentThread();
            StackTraceElement[] stack = t.getStackTrace();
            StackTraceElement caller = stack[2];
            return caller.getMethodName();
        }
        @Test
        @DisplayName("Confirm that deserializing a known subclass works")
        public void test3() throws JsonProcessingException {
            String jsonString =
                    "{ \"strings\":[\"Alan\", \"Ellie\", \"Ian\"]," +
                            "\"integerList\":[1,2,3,4,5,6]}";
            ObjectMapper objectMapper = new ObjectMapper();
            JsonNode node = objectMapper.readTree(jsonString);
            assertNotNull(node);
            assertNotNull(getCurrentStackFrame());
        }
        @Test
        @DisplayName("Check how mixed collections work")
        public void test4() throws JsonProcessingException {
            String jsonString = makeJSONForMessage3();
            ObjectMapper objectMapper = new ObjectMapper();
            Message3 msg3 = objectMapper.readValue(jsonString, Message3.class);
            Assertions.assertNotNull(msg3);
            assertEquals(NUM_MSGS, msg3.getMsgs().size());
            int i = 0;
            for (Message1 m : msg3.getMsgs()) {
               if (i % 2 == 0) {
                   Assertions.assertTrue(m instanceof Message2);
               } else {
                   Assertions.assertFalse(m instanceof Message2);
               }
```

```
                ++i;
            }
        }
        @Test
        @DisplayName("Play with MyObjectMapper")
        public void test5() throws JsonProcessingException {
            String jsonString = makeJSONForMessage3();
            ObjectMapper objectMapper = new ObjectMapper();
            Message3 msg3 = objectMapper.readValue(jsonString, Message3.class);
            var tf = objectMapper.getTypeFactory();
            tf.constructFromCanonical(Message2.class.getCanonicalName());
            tf.constructFromCanonical(Message1.class.getCanonicalName());
            Assertions.assertNotNull(msg3);
            assertEquals(NUM_MSGS, msg3.getMsgs().size());
            boolean foundMsg2 = false;
            for (Message1 m : msg3.getMsgs()) {
                if (m instanceof Message2) foundMsg2 = true;
            }
            Assertions.assertTrue(foundMsg2);
        }
        @Test
        @DisplayName("Test Lombok builder behavior")
        public void test100() {
            Message2.builder()
                    .ints(List.of(1, 2, 3, 4, 5, 6))
                    .strings(List.of("Alan", "Ellie", "Ian"));
        }
    }
    class FailureTests {
    }
}
```

A.318 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/json/WrapperTest.java

```
package mil.sstaf.core.json;

import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.node.ObjectNode;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Nested;
import org.junit.jupiter.api.Test;

import java.nio.file.Path;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;

import static org.junit.jupiter.api.Assertions.*;

public class WrapperTest {

    @Nested
    @DisplayName("Test the 'Happy Paths'")
    class HappyTests {
        @Test
        @DisplayName("Confirm that reference counting works")
        public void testRefCount1() {
            ObjectMapperFactory omf = path -> null;
            Map<String, JsonNode> cache = new HashMap<>();

            ReferenceWrapper one = new ReferenceWrapper("bob", null, null, omf, cache);
            ReferenceWrapper two = new ReferenceWrapper("bob", one, null, omf, cache);
            ReferenceWrapper three = new ReferenceWrapper("diane", two, null, omf, cache);
            ReferenceWrapper four = new ReferenceWrapper("bob", three, null, omf, cache);

            assertEquals(3, four.refCount("bob"));
            assertEquals(1, four.refCount("diane"));
            assertEquals(0, four.refCount("luigi"));
        }

        @Test
        @DisplayName("Simplest test to confirm that a ReferenceWrapper can resolve a reference")
        void resolveTest1() {
            assertDoesNotThrow(() -> {
                Map<String, JsonNode> referenceCache = new HashMap<>();
                ObjectMapperFactory omf = path -> new ObjectMapper();

                //
                // Create the JsonNode that will be inserted into the graph.
                // Stick it in the cache.
                //
                String jsonString1 = "{\"class\":\"mil.sstaf.core.json.Message2\", \"ints\":[1,2,
3,4,5,6]}";
                JsonNode node1 = omf.create(null).readTree(jsonString1);
                String referenceName = "/path/to/file.json";
                referenceCache.put(referenceName, node1);

                //
                // Create the top-level object that uses a reference
                //
                String jsonString2 = "{\"it\":\"/path/to/file.json\"}";
                JsonNode topNode = omf.create(null).readTree(jsonString2);
                JsonNode itNode = topNode.get("it");
                assertEquals(referenceName, itNode.asText());
```

```
                //
                // Build the wrappers
                //
                ObjectNodeWrapper topWrapper = new ObjectNodeWrapper(null,
                        (ObjectNode) topNode, omf, referenceCache);
                ReferenceWrapper wrapper = new ReferenceWrapper(
                        itNode.asText(), topWrapper, itNode, omf, referenceCache);

                //
                // Resolve the reference and confirm
                //
                wrapper.resolve();
                assertTrue(wrapper.resolved);
                JsonNode replaced = topNode.get("it");
                assertEquals(node1, replaced);
            });
        }

        @Test
        @DisplayName("Try invoking resolution from the top level")
        void resolveTest2() {
            assertDoesNotThrow(() -> {
                Map<String, JsonNode> referenceCache = new HashMap<>();
                ObjectMapperFactory omf = path -> new ObjectMapper();

                //
                // Create the JsonNode that will be inserted into the graph.
                // Stick it in the cache.
                //
                String jsonString1 = "{\"class\":\"mil.sstaf.core.json.Message2\", \"ints\":[1,2,
3,4,5,6]}";
                JsonNode node1 = omf.create(null).readTree(jsonString1);
                String referenceName = "/path/to/file.json";
                referenceCache.put(referenceName, node1);

                //
                // Create the top-level object that uses a reference
                //
                String jsonString2 = "{\"it\":\"/path/to/file.json\"}";
                JsonNode topNode = omf.create(null).readTree(jsonString2);
                JsonNode itNode = topNode.get("it");
                assertEquals(referenceName, itNode.asText());

                //
                // Build the top-level wrappers
                //
                ObjectNodeWrapper topWrapper = new ObjectNodeWrapper(null,
                        (ObjectNode) topNode, omf, referenceCache);

                //
                // Resolve the reference and confirm
                //
                topWrapper.resolve();
                assertTrue(topWrapper.resolved);
                JsonNode replaced = topNode.get("it");
                assertEquals(node1, replaced);
            });
        }

        @Test
        @DisplayName("Confirm that deserialization works for two-node system.")
        void resolveTest3() {
            assertDoesNotThrow(() -> {
                Map<String, JsonNode> referenceCache = new HashMap<>();
                ObjectMapperFactory omf = path -> new ObjectMapper();

                //
```

```
                // Create the JsonNode that will be inserted into the graph.
                // Stick it in the cache.
                //
                String jsonString1 = "{\"class\":\"mil.sstaf.core.json.Message2\", \"ints\":[1,2,
3,4,5,6]}";
                JsonNode node1 = omf.create(null).readTree(jsonString1);
                String referenceName = "/path/to/file.json";
                referenceCache.put(referenceName, node1);

                //
                // Create the top-level object that uses a reference
                //
                String jsonString2 = "{\"class\":\"mil.sstaf.core.json.Message4\"," +
                        " \"it\":\"/path/to/file.json\"}";
                JsonNode topNode = omf.create(null).readTree(jsonString2);
                JsonNode itNode = topNode.get("it");
                assertEquals(referenceName, itNode.asText());

                //
                // Build the top-level wrappers
                //
                ObjectNodeWrapper topWrapper = new ObjectNodeWrapper(null,
                        (ObjectNode) topNode, omf, referenceCache);

                //
                // Resolve the reference and confirm
                //
                topWrapper.resolve();
                assertTrue(topWrapper.resolved);
                JsonNode replaced = topNode.get("it");
                assertEquals(node1, replaced);
                ObjectMapper bob = omf.create(null);
                Message4 message4 = bob.treeToValue(topNode, Message4.class);
                assertNotNull(message4);
                assertNotNull(message4.getIt());
                Message2 message2 = message4.getIt();
                assertEquals(6, message2.getInts().size());
            });
        }

        @Test
        @DisplayName("Confirm that deserialization works for file-based, three-node system.")
        void resolveTest4() {
            assertDoesNotThrow(() -> {
                Map<String, JsonNode> referenceCache = new ConcurrentHashMap<>();
                ObjectMapperFactory omf = path -> new ObjectMapper();

                String userDir = System.getProperty("user.dir");
                Path basePath = Path.of(userDir, "src/test/resources/jsonTests");

                Path topFile = Path.of(basePath.toString(), "message5.json");
                JsonNode topNode = omf.create(null).readTree(topFile.toFile());
                ObjectNodeWrapper topWrapper = new ObjectNodeWrapper(null,
                        (ObjectNode) topNode, omf, referenceCache);
                topWrapper.setDirectory(basePath);

                //
                // Resolve the reference and confirm
                //
                topWrapper.resolve();
                assertTrue(topWrapper.resolved);
                ObjectMapper bob = omf.create(null);
                Message5 message5 = bob.treeToValue(topNode, Message5.class);
                assertNotNull(message5);
                assertNotNull(message5.getMsg4());
                Message2 message2 = message5.getMsg4().getIt();
                assertEquals(6, message2.getInts().size());
            });
```

```
        }

        @Test
        @DisplayName("Confirm that deserialization works for arrays of references.")
        void resolveTest5() {
            assertDoesNotThrow(() -> {
                Map<String, JsonNode> referenceCache = new ConcurrentHashMap<>();
                ObjectMapperFactory omf = path -> new ObjectMapper();

                String userDir = System.getProperty("user.dir");
                Path basePath = Path.of(userDir, "src/test/resources/jsonTests");

                Path topFile = Path.of(basePath.toString(), "message6.json");
                JsonNode topNode = omf.create(null).readTree(topFile.toFile());
                ObjectNodeWrapper topWrapper = new ObjectNodeWrapper(null,
                        (ObjectNode) topNode, omf, referenceCache);
                topWrapper.setDirectory(basePath);

                //
                // Resolve the reference and confirm
                //
                topWrapper.resolve();
                assertEquals(2, referenceCache.size());
                assertTrue(topWrapper.resolved);
                ObjectMapper bob = omf.create(null);
                Message6 message6 = bob.treeToValue(topNode, Message6.class);
                assertNotNull(message6);
                assertNotNull(message6.getFours());
                List<Message4> fours = message6.getFours();
                assertEquals(4, fours.size());
                for (Message4 msg4 : fours) {
                    assertNotNull(msg4.getIt());
                    Message2 msg2 = msg4.getIt();
                    assertEquals(6, msg2.getInts().size());
                }
            });
        }
    }

    @Nested
    @DisplayName("Test scenarios that cause failures")
    class FailureTests {
        @Test
        @DisplayName("Confirm that circular references throw an exception.")
        void resolveTest5() {
            JsonResolutionException jre = assertThrows(JsonResolutionException.class, () -> {
                Map<String, JsonNode> referenceCache = new ConcurrentHashMap<>();
                ObjectMapperFactory omf = path -> new ObjectMapper();

                String userDir = System.getProperty("user.dir");
                Path basePath = Path.of(userDir, "src/test/resources/jsonTests");

                Path topFile = Path.of(basePath.toString(), "message8.json");
                JsonNode topNode = omf.create(null).readTree(topFile.toFile());
                ObjectNodeWrapper topWrapper = new ObjectNodeWrapper(null,
                        (ObjectNode) topNode, omf, referenceCache);
                topWrapper.setDirectory(basePath);

                //
                // Resolve the reference and confirm
                //
                topWrapper.resolve();
            });
            assertTrue(jre.getMessage().contains("Circular"));
        }


        @Test
```

```
        @DisplayName("Confirm that bad references throw an exception.")
        void resolveTest6() {
            JsonResolutionException jre = assertThrows(JsonResolutionException.class, () -> {
                Map<String, JsonNode> referenceCache = new ConcurrentHashMap<>();
                ObjectMapperFactory omf = path -> new ObjectMapper();

                String userDir = System.getProperty("user.dir");
                Path basePath = Path.of(userDir, "src/test/resources/jsonTests");

                Path topFile = Path.of(basePath.toString(), "badref.json");
                JsonNode topNode = omf.create(null).readTree(topFile.toFile());
                ObjectNodeWrapper topWrapper = new ObjectNodeWrapper(null,
                        (ObjectNode) topNode, omf, referenceCache);
                topWrapper.setDirectory(basePath);

                //
                // Resolve the reference and confirm
                //
                topWrapper.resolve();
            });
            assertTrue(jre.getMessage().contains("Could not read reference"));
            assertTrue(jre.getMessage().contains("not_there.json"));
        }
    }
}
```

A.319 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/mocks/BadMessage1.java

```
package mil.sstaf.core.mocks;

import lombok.Builder;
import lombok.extern.jackson.Jacksonized;

@Builder
@Jacksonized
public class BadMessage1 {

    public final int one;
    public final int two;

}
```

A.320 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/mocks/BadMessage2.java

```
package mil.sstaf.core.mocks;


import lombok.Builder;
import lombok.extern.jackson.Jacksonized;

@Builder
@Jacksonized
public final class BadMessage2 {

    public final int one;
    public final int two;

}
```

A.321 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/mocks/BadMessage3.java

```
package mil.sstaf.core.mocks;

import lombok.Builder;
import lombok.extern.jackson.Jacksonized;

@Builder
@Jacksonized
public final
class BadMessage3 {
    public final int one;
    public final int two;
}
```

A.322 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/mocks/BadMessage4.java

```
package mil.sstaf.core.mocks;

import lombok.Builder;
import lombok.extern.jackson.Jacksonized;

@Builder
@Jacksonized
public class BadMessage4 {
    public final int one;
    public final int two;
}
```

A.323 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/mocks/BlobContent.java

```
package mil.sstaf.core.mocks;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.EqualsAndHashCode;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import mil.sstaf.core.features.HandlerContent;
@SuperBuilder
@Jacksonized
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
@EqualsAndHashCode(callSuper = true)
public class BlobContent extends HandlerContent {
    private final byte[] blob;
}
```

A.324 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/mocks/Brain.java

```
package mil.sstaf.core.mocks;
import mil.sstaf.core.features.BaseFeature;
import mil.sstaf.core.features.FeatureConfiguration;
public class Brain extends BaseFeature implements BrainProvider {
    public Brain() {
        super("Brain", 314, 0, 0, false, "Are you pondering what I am pondering?");
    }

    @Override
    public Class<? extends FeatureConfiguration> getConfigurationClass() {
        return FeatureConfiguration.class;
    }
}
```

A.325 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/mocks/BrainProvider.java

```
package mil.sstaf.core.mocks;
import mil.sstaf.core.features.Feature;
public interface BrainProvider extends Feature {
}
```

A.326 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/mocks/Command1.java

```
package mil.sstaf.core.mocks;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.EqualsAndHashCode;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import mil.sstaf.core.features.HandlerContent;
@SuperBuilder
@Jacksonized
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
@EqualsAndHashCode(callSuper = true)
public class Command1 extends HandlerContent {
    public String string;
    public boolean done;
    public Command1() {
        string = "Waiting";
        done = false;
    }
}
```

A.327 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/mocks/Handler1.java

```
package mil.sstaf.core.mocks;
import mil.sstaf.core.entity.Address;
import mil.sstaf.core.entity.Message;
import mil.sstaf.core.features.*;
import mil.sstaf.core.util.Injected;
import mil.sstaf.core.util.SSTAFException;
import java.util.List;
public class Handler1 extends BaseHandler {
    @Injected
    public String myString;
    @Injected(name = "+")
    public Long posLong;
    @Injected(name = "-")
    public Long negLong;
    @Requires(name = "Pinky")
    public PinkyProvider pinky = null;
    public Handler1() {
        super("Handler1", 3, 1, 7, false,
                "Handle it Roy. Handle it, handle it");
        myString = null;
        posLong = null;
        negLong = null;
    }
    @Override
    public List<Class<? extends HandlerContent>> contentHandled() {
        return List.of(Command1.class, StringContent.class);
    }
    @Override
    public void init() {
        super.init();
        if (pinky == null) {
            throw new SSTAFException("Narf!");
        }
    }
    @Override
    public Class<? extends FeatureConfiguration> getConfigurationClass() {
        return FeatureConfiguration.class;
    }
    @Override
    public ProcessingResult process(HandlerContent arg, long scheduledTime_ms, long currentTime_m
s, Address from, long id, Address respondTo) {
        if (arg instanceof Command1) {
            Command1 tc1 =  Command1.builder().done(true).string("Got it").build();
            Message result = this.buildNormalResponse(tc1, id, respondTo);
            return ProcessingResult.of(result);
        } else {
            return ProcessingResult.empty();
        }
    }
}
```

A.328 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/mocks/Handler2.java

```
package mil.sstaf.core.mocks;
import mil.sstaf.core.entity.Address;
import mil.sstaf.core.features.*;
import mil.sstaf.core.util.Injected;
import mil.sstaf.core.util.SSTAFException;
import java.util.List;
import java.util.UUID;
public class Handler2 extends BaseHandler {
    @Injected
    public UUID uuid;
    @Injected
    public String string;
    @Requires(name = "Brain")
    public BrainProvider brain;
    public Handler2() {
        super("Handler2", 0, 0, 0, false,
                "This is the second handler");
    }
    @Override
    public List<Class<? extends HandlerContent>> contentHandled() {
        return List.of(IntContent.class, LongContent.class);
    }
    @Override
    public void init() {
        super.init();
        if (brain == null) {
            throw new SSTAFException("Zorp! Brain has not been injected.");
        }
    }
    @Override
    public Class<? extends FeatureConfiguration> getConfigurationClass() {
        return FeatureConfiguration.class;
    }
    @Override
    public ProcessingResult process(HandlerContent arg, long scheduledTime_ms, long currentTime_m
s, Address from, long id, Address respondTo) {
        return null;
    }
}
```

A.329 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/mocks/Handler3.java

```
package mil.sstaf.core.mocks;
import mil.sstaf.core.entity.Address;
import mil.sstaf.core.features.*;
import java.util.List;
import java.util.Objects;
public class Handler3 extends BaseHandler {

    @Requires(name = "Handler2")
    Handler2 handler2;
    @Requires(name = "Handler1")
    Handler1 handler1;
    public Handler3() {
        super("Handler3", 0, 1, 0, false,
                "Three shall be the number of the Handlers");
    }
    @Override
    public Class<? extends FeatureConfiguration> getConfigurationClass() {
        return FeatureConfiguration.class;
    }
    @Override
    public List<Class<? extends HandlerContent>> contentHandled() {
        return List.of(BlobContent.class);
    }
    @Override
    public void init() {
        super.init();
        Objects.requireNonNull(handler1);
        Objects.requireNonNull(handler2);
    }
    @Override
    public ProcessingResult process(HandlerContent arg, long scheduledTime_ms, long currentTime_m
s, Address from, long id, Address respondTo) {
        return null;
    }
}
```

A.330 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/mocks/Pinky.java

```
package mil.sstaf.core.mocks;
import mil.sstaf.core.features.BaseFeature;
import mil.sstaf.core.features.FeatureConfiguration;
public class Pinky extends BaseFeature implements PinkyProvider {

    public Pinky() {
        super("Pinky", 13, 0, 0, false, "NARF!");
    }
    @Override
    public Class<? extends FeatureConfiguration> getConfigurationClass() {
        return FeatureConfiguration.class;
    }
}
```

A.331 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/mocks/PinkyProvider.java

```
package mil.sstaf.core.mocks;
import mil.sstaf.core.features.Feature;
public interface PinkyProvider extends Feature {
}
```

A.332 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/mocks/SillyMessage.java

```
package mil.sstaf.core.mocks;

import lombok.Builder;
import lombok.extern.jackson.Jacksonized;

@Builder
@Jacksonized
public final class SillyMessage {

    public final int one;
    public final int two;


}
```

A.333 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/mocks/SnowballProvider.java

```
package mil.sstaf.core.mocks;
import mil.sstaf.core.features.Feature;
public interface SnowballProvider extends Feature {
}
```

A.334 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/mocks/UnsupportedCommand.java

```
package mil.sstaf.core.mocks;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.EqualsAndHashCode;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import mil.sstaf.core.features.HandlerContent;
@SuperBuilder
@Jacksonized
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
@EqualsAndHashCode(callSuper = true)
public class UnsupportedCommand extends HandlerContent {
}
```

A.335 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/util/CompletionServiceAdapterTest.java

```
package mil.sstaf.core.util;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
class CompletionServiceAdapterTest {
    public static final int NUM_VALUES = 2345;
    @Test
    void testCSA() {
        ExecutorService executorService = Executors.newFixedThreadPool(16);
        Random r = new Random(NUM_VALUES);
        List<Callable<Long>> callables = new ArrayList<>(NUM_VALUES);
        Accumulator accumulatorOrig = new Accumulator();
        for (int i = 0; i < NUM_VALUES; ++i) {
            Task t = new Task();
            t.value = Math.abs(r.nextInt());
            callables.add(t);
            accumulatorOrig.add(t.value);
        }
        Accumulator accumulator = new Accumulator();
        CompletionServiceAdapter.processCompletions(callables, executorService, accumulator::add)
;
        Assertions.assertEquals(accumulatorOrig.total, accumulator.total);
    }
    static class Accumulator {
        long total = 0;
        void add(long value) {
            total += value;
        }
    }
    static class Task implements Callable<Long> {
        long value = 0;
        @Override
        public Long call() throws Exception {
            Thread.sleep(value % 17);
            return value;
        }
    }
}
```

A.336 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/util/FormatterTest.java

```
package mil.sstaf.core.util;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;
public class FormatterTest {
    @Test
    void formatterWorks() {
        long val = 1;
        Assertions.assertEquals("0:0:0:0.001", Formatter.millisToDHMS(val));
        val = 1456;
        Assertions.assertEquals("0:0:0:1.456", Formatter.millisToDHMS(val));
        val = 17456 + 32000 * 60;
        Assertions.assertEquals("0:0:32:17.456", Formatter.millisToDHMS(val));
        val = 17456 + 32000 * 60 + 15 * 3600 * 1000;
        Assertions.assertEquals("0:15:32:17.456", Formatter.millisToDHMS(val));
        val = 17456 + 32000 * 60 + 15 * 3600 * 1000L + 95 * 24 * 3600 * 1000L;
        Assertions.assertEquals("95:15:32:17.456", Formatter.millisToDHMS(val));
    }
}
```

A.337 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/util/InjectorTest.java

```
package mil.sstaf.core.util;
import org.junit.jupiter.api.Test;
import java.util.ArrayDeque;
import java.util.Queue;
import java.util.UUID;
import java.util.concurrent.LinkedBlockingDeque;
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertNull;
class InjectorTest {
    @Test
    void canInjectArbitraryObjectsByType() {
        InjectionTarget injectionTarget = new InjectionTarget();
        Long lv = 3141592L;
        String sv = "I am the very model of a modern Major General";
        UUID uv = UUID.randomUUID();
        assertNull(injectionTarget.longValue);
        assertNull(injectionTarget.stringValue);
        assertNull(injectionTarget.uuid);
        Injector.inject(injectionTarget, lv);
        Injector.inject(injectionTarget, sv);
        Injector.inject(injectionTarget, uv);
        assertEquals(lv, injectionTarget.longValue);
        assertEquals(sv, injectionTarget.stringValue);
        assertEquals(uv, injectionTarget.uuid);
    }
    @Test
    void canInjectObjectsUsingVarargs() {
        InjectionTarget injectionTarget = new InjectionTarget();
        Long lv = 3141592L;
        String sv = "I am the very model of a modern Major General";
        UUID uv = UUID.randomUUID();

        assertNull(injectionTarget.longValue);
        assertNull(injectionTarget.stringValue);
        assertNull(injectionTarget.uuid);
        Injector.injectAll(injectionTarget, lv, sv, uv);
        assertEquals(lv, injectionTarget.longValue);
        assertEquals(sv, injectionTarget.stringValue);
        assertEquals(uv, injectionTarget.uuid);
    }
    @Test
    void canSpecifyNames() {
        NamedTarget target = new NamedTarget();
        assertNull(target.in);
        assertNull(target.out);
        Queue<Object> iv = new LinkedBlockingDeque<>();
        Queue<Object> ov = new ArrayDeque<>();
        Injector.inject(target, "inQueue", iv);
        Injector.inject(target, "outQueue", ov);
        assertEquals(iv, target.in);
        assertEquals(ov, target.out);
    }
    private static class InjectionTarget {
        @Injected
        private final Long longValue = null;
        @Injected
        private final String stringValue = null;
        @Injected
        private final UUID uuid = null;
    }
    private static class NamedTarget {
        @Injected(name = "inQueue")
```

```
        Queue<Object> in = null;
        @Injected(name = "outQueue")
        Queue<Object> out = null;
    }
}
```

A.338 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/util/Message1.java

```
package mil.sstaf.core.util;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.Builder;
import lombok.Getter;
import lombok.extern.jackson.Jacksonized;
@Builder
@Jacksonized
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
public class Message1 {
    @Getter
    private String name;
    @Getter
    private int value;
}
```

A.339 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/util/Message2.java

```
package mil.sstaf.core.util;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.Builder;
import lombok.extern.jackson.Jacksonized;
import java.util.List;
@Builder
@Jacksonized
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
public final class Message2 {
    final String tag;
    final List<Message1> message1List;
}
```

A.340 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/util/PairValueMapTest.java

```
package mil.sstaf.core.util;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
public class PairValueMapTest {
    @Test
    void reversingOrderOfKeysPreservesEquals() {
        PairValueMap<Integer, Integer> map = new PairValueMap<>();
        PairValueMap.KeyPair<Integer> key1 = new PairValueMap.KeyPair<>(100, 4301);
        PairValueMap.KeyPair<Integer> key2 = new PairValueMap.KeyPair<>(4301, 100);
        PairValueMap.KeyPair<Integer> key3 = new PairValueMap.KeyPair<>(17, 1701);
        PairValueMap.KeyPair<Integer> key4 = new PairValueMap.KeyPair<>(1701, 17);
        assertEquals(key1, key1);
        assertEquals(key2, key2);
        assertEquals(key1, key2);
        assertNotEquals(key1, key3);
        assertNotEquals(key1, key4);
        assertNotEquals(key2, key3);
        assertNotEquals(key2, key4);
        assertEquals(key3, key4);
        assertNotEquals(key1, 15);
    }
    @Test
    void reversingOrderOfKeysPreservesHash() {
        PairValueMap<Integer, Integer> map = new PairValueMap<>();
        PairValueMap.KeyPair<Integer> key1 = new PairValueMap.KeyPair<>(100, 4301);
        PairValueMap.KeyPair<Integer> key2 = new PairValueMap.KeyPair<>(4301, 100);
        PairValueMap.KeyPair<Integer> key3 = new PairValueMap.KeyPair<>(17, 1701);
        PairValueMap.KeyPair<Integer> key4 = new PairValueMap.KeyPair<>(1701, 17);
        assertEquals(key1.hashCode(), key2.hashCode());
        assertEquals(key3.hashCode(), key4.hashCode());
    }
    @Test
    void putsAndGetsWork() {
        PairValueMap<Integer, Integer> map = new PairValueMap<>();
        PairValueMap.KeyPair<Integer> key1 = new PairValueMap.KeyPair<>(100, 4301);
        PairValueMap.KeyPair<Integer> key2 = new PairValueMap.KeyPair<>(4301, 100);
        PairValueMap.KeyPair<Integer> key3 = new PairValueMap.KeyPair<>(17, 1701);
        PairValueMap.KeyPair<Integer> key4 = new PairValueMap.KeyPair<>(1701, 17);
        Integer value1 = 3333;
        Integer value2 = 6666;
        map.put(key1, value1);
        assertNull(map.get(key3));
        assertNull(map.get(key4));
        map.put(key3, value2);
        Integer retrieved1 = map.get(key2);
        Integer retrieved2 = map.get(key4);
        assertNotNull(retrieved1);
        assertNotNull(retrieved2);
        assertEquals(value1, retrieved1);
        assertEquals(value2, retrieved2);
        map.remove(key1);
        assertNull(map.get(key2));
        assertNull(map.get(key1));
    }
}
```

A.341 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/util/ReflectionUtilsTest.java

```
package mil.sstaf.core.util;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
import java.beans.BeanProperty;
import java.lang.reflect.Field;
import java.lang.reflect.Method;
import java.util.List;
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertTrue;
public class ReflectionUtilsTest {
    @Test
    @DisplayName("Confirm that a method annotated with a specific Annotation can be found")
    void test1() {
        class Oingo {
            private final int bob = 14;
            @Deprecated
            public int getBob() {
                return bob;
            }
        }
        class Boingo {
            public Object getIt() {
                return null;
            }
            public Object getItToo() {
                return null;
            }
        }
        List<Method> methodList = ReflectionUtils.getMethodsWithAnnotation(Oingo.class, Deprecate
d.class);
        assertEquals(1, methodList.size());
        assertEquals("getBob", methodList.get(0).getName());
        methodList = ReflectionUtils.getMethodsWithAnnotation(Boingo.class, Deprecated.class);
        assertEquals(0, methodList.size());
    }
    @Test
    @DisplayName("Confirm that if a required annotation is not found a SSTAFException is thrown."
)
    void test2() {
        class Boingo {
            @BeanProperty
            public Object getIt() {
                return null;
            }
            @BeanProperty
            public Object getItToo() {
                return null;
            }
        }
        Assertions.assertDoesNotThrow(() -> {
            List<Method> methodList = ReflectionUtils.getMethodsWithAnnotationOrThrow(Boingo.clas
s, BeanProperty.class);
            assertEquals(2, methodList.size());
            assertTrue(methodList.stream().anyMatch(it -> it.getName().matches("getIt")));
            assertTrue(methodList.stream().anyMatch(it -> it.getName().matches("getItToo")));
        });
        SSTAFException boom = Assertions.assertThrows(SSTAFException.class, () -> {
            List<Method> methodList = ReflectionUtils.getMethodsWithAnnotationOrThrow(Boingo.clas
s,
                    Deprecated.class);
            assertEquals(0, methodList.size());
```

```
        });
        assertTrue(boom.getMessage().contains("Deprecated"));
        assertTrue(boom.getMessage().contains("Boingo"));
    }
    @Test
    @DisplayName("Confirm that getAllFields() gets all fields in a hierarchy")
    void test3() {
        List<Field> fields = ReflectionUtils.getAllFields(C.class);
        Assertions.assertEquals(5, fields.size());
        boolean haveBob = false;
        boolean haveSam = false;
        boolean haveEmma = false;
        boolean haveEarl = false;
        boolean haveTheList = false;
        for (Field f : fields) {
            if (f.getName().equals("bob")) haveBob = true;
            if (f.getName().equals("sam")) haveSam = true;
            if (f.getName().equals("emma")) haveEmma = true;
            if (f.getName().equals("earl")) haveEarl = true;
            if (f.getName().equals("theList")) haveTheList = true;
        }
        assertTrue(haveBob);
        assertTrue(haveSam);
        assertTrue(haveEmma);
        assertTrue(haveEarl);
        assertTrue(haveTheList);
    }
    static class A {
        private Integer bob;
        private String sam;
    }
    static class B extends A {
        private Double emma;
        private Long earl;
    }
    static class C extends B {
        private List<String> theList;
    }
}
```

A.342 SSTAF/src/framework/mil.sstaf.core/src/test/java/mil/sstaf/core/util/ValidationTest.java

```
package mil.sstaf.core.util;
import org.junit.jupiter.api.Test;
import java.util.function.Predicate;
import static mil.sstaf.core.util.Validation.require;
import static org.junit.jupiter.api.Assertions.*;
public class ValidationTest {
    @Test
    void inlinePredicatesWork() {
        assertThrows(IllegalArgumentException.class, () -> require(-3, val -> val > 0, "Ack!"));
        assertThrows(IllegalArgumentException.class, () -> require(-3, val -> val > 0));
        assertEquals(3, require(3, val -> val > 0));
        assertEquals(3, require(3, val -> val > 0, "This is bad"));
    }
    @Test
    void declaredPredicateWorks() {
        Predicate<Object> predicate = o -> o instanceof String;
        assertThrows(IllegalArgumentException.class, () ->
                require(4, predicate));
        assertDoesNotThrow(() ->
                require("I am a string", predicate));
    }
}
```

A.343 SSTAF/src/framework/mil.sstaf.core/src/test/resources/HumanFactoryTest/BadHuman1.json

```
{
  "name": "Test Dude",
  "features": {
  },
  "configurations": {
    "Simple Kinematics": {}
  }
}
```

A.344 SSTAF/src/framework/mil.sstaf.core/src/test/resources/HumanFactoryTest/BadHuman2.json

```
{
  "name": "Test Dude",
  "features": []
}
```

A.345 SSTAF/src/framework/mil.sstaf.core/src/test/resources/HumanFactoryTest/BadHuman3.json

```
{
  "name": "Test Dude",
  "configurations": {}
}
```

A.346 SSTAF/src/framework/mil.sstaf.core/src/test/resources/HumanFactoryTest/BadHuman4.json

```
{
  "name": "Test Dude",
  "anthro": {
    "sex": "Female",
    "age": 23,
    "height_cm": 150,
    "span_cm": 150,
    "weight_kg": 50.0
  },
  "kinematicsProvider": {
    "serviceName": "Santos",
    "majorVersion": 3,
    "minorVersion": 1,
    "requireExact": true,
    "configuration": {}
  },
  "features": [],
  "configurations": {}
}
```

A.347 SSTAF/src/framework/mil.sstaf.core/src/test/resources/HumanFactoryTest/FannyPack.json

```
{
  "name": "Fanny Pack",
  "mass_kg": 1.5
}
```

A.348 SSTAF/src/framework/mil.sstaf.core/src/test/resources/HumanFactoryTest/TestHuman1.json

```
{
  "class" : "mil.sstaf.core.entity.Human",
  "name": "Test Dude",
  "features": [],
  "configurations": {}
}
```

A.349 SSTAF/src/framework/mil.sstaf.core/src/test/resources/HumanFactoryTest/TestHuman2.json

```
{
  "class" : "mil.sstaf.core.entity.Human",
  "name": "Test Dude",
  "anthro": {
    "class" : "mil.sstaf.core.features.FeatureConfiguration",
    "sex": "Female",
    "age": 23,
    "height_cm": 150,
    "span_cm": 150,
    "weight_kg": 50.0
  },
  "mil.kinematicsProvider": {
    "class" : "mil.sstaf.core.features.FeatureConfiguration",
    "serviceName": "Santos",
    "majorVersion": 3,
    "minorVersion": 1,
    "requireExact": true,
    "configuration": {}
  },
  "injuryProvider": "ip.json",
  "features": [],
  "configurations": {}
}
```

A.350 SSTAF/src/framework/mil.sstaf.core/src/test/resources/HumanFactoryTest/anthro.json

```
{
  "class" : "mil.sstaf.core.features.FeatureConfiguration",
  "sex": "Female",
  "age": 23,
  "height_cm": 150,
  "span_cm": 150,
  "weight_kg": 50
}
```

A.351 SSTAF/src/framework/mil.sstaf.core/src/test/resources/HumanFactoryTest/ip.json

```
{
  "class" : "mil.sstaf.core.features.FeatureConfiguration",
  "serviceName": "ORCA",
  "majorVersion": 3,
  "minorVersion": 1,
  "requireExact": true,
  "configuration": {}
}
```

A.352 SSTAF/src/framework/mil.sstaf.core/src/test/resources/HumanFactoryTest/kp.json

```
{
  "serviceName": "Santos",
  "majorVersion": 3,
  "minorVersion": 1,
  "requireExact": true,
  "configuration": {}
}
```

A.353 SSTAF/src/framework/mil.sstaf.core/src/test/resources/SoldierFactoryTest/BadSoldier1.json

```
{
  "class" : "mil.sstaf.core.entity.Soldier",
  "name": "Test Dude",
  "anthro": {
    "sex": "Female",
    "age": 23,
    "height_cm": 150,
    "span_cm": 150,
    "weight_kg": 50.0
  },
  "kinematicsProvider": {
    "serviceName": "Santos",
    "majorVersion": [],
    "minorVersion": 1,
    "requireExact": true,
    "configuration": {}
  },
  "rank": "E6",
  "kit": "TestKit.json",
  "injuryProvider": "src/test/resources/humanFactoryTest/ip.json",
  "aimProvider": "ap.json",
  "targetingProvider": "tp.json"
}
```

A.354 SSTAF/src/framework/mil.sstaf.core/src/test/resources/SoldierFactoryTest/BadSoldier2.json

```
{
  "class" : "mil.sstaf.core.entity.Soldier",
  "name": "Test Dude",
  "kinematicsProvider": {
    "serviceName": "Santos",
    "majorVersion": [],
    "minorVersion": 1,
    "requireExact": true,
    "configuration": {}
  },
  "rank": "E6",
  "kit": "TestKit.json",
  "injuryProvider": "src/test/resources/humanFactoryTest/ip.json",
  "aimProvider": "ap.json",
  "targetingProvider": "tp.json"
}
```

A.355 SSTAF/src/framework/mil.sstaf.core/src/test/resources/SoldierFactoryTest/BadSoldier3.json

```
{
  "name": "Test Dude",
  "anthro": {
    "sex": "Female",
    "age": 23,
    "height_cm": 150,
    "span_cm": 150,
    "weight_kg": 50.0
  },
  "rank": "E6",
  "kit": "TestKit.json",
  "injuryProvider": "../humanFactoryTest/ip.json",
  "aimProvider": "ap.json",
  "targetingProvider": "tp.json"
}
```

A.356 SSTAF/src/framework/mil.sstaf.core/src/test/resources/SoldierFactoryTest/BadSoldier4.json

```
{
  "name": "Test Dude",
  "anthro": {
    "sex": "Female",
    "age": 23,
    "height_cm": 150,
    "span_cm": 150,
    "weight_kg": 50.0
  },
  "kinematicsProvider": {
    "serviceName": "Santos",
    "majorVersion": 3,
    "minorVersion": 1,
    "requireExact": true,
    "configuration": {}
  },
  "rank": "E6",
  "kit": "TestKit.json",
  "aimProvider": "ap.json",
  "targetingProvider": "tp.json"
}
```

A.357 SSTAF/src/framework/mil.sstaf.core/src/test/resources/SoldierFactoryTest/BadSoldier5.json

```
{
  "name": "Test Dude",
  "anthro": "anthro.json",
  "rank": "E6",
  "kit": "TestKit.json",
  "kinematicsProvider": "kp.json",
  "injuryProvider": "ip.json",
  "targetingProvider": "tp.json"
}
```

A.358 SSTAF/src/framework/mil.sstaf.core/src/test/resources/SoldierFactoryTest/BadSoldier6.json

```
{
  "name": "Test Dude",
  "anthro": "anthro.json",
  "rank": "E6",
  "kit": "TestKit.json",
  "kinematicsProvider": "kp.json",
  "injuryProvider": "ip.json",
  "aimProvider": "ap.json"
}
```

A.359 SSTAF/src/framework/mil.sstaf.core/src/test/resources/SoldierFactoryTest/FannyPack.json

```
{
  "name": "Fanny Pack",
  "mass_kg": 1.5
}
```

A.360 SSTAF/src/framework/mil.sstaf.core/src/test/resources/SoldierFactoryTest/M16A1.json

```
{
  "name": "M16A1",
  "magazineType": "5.56mm STANAG",
  "emptyMass_kg": 5.5
}
```

A.361 SSTAF/src/framework/mil.sstaf.core/src/test/resources/SoldierFactoryTest/STANAGMagazine.json

```
{
  "name": "5.56mm STANAG 30rd",
  "magazineType": "5.56mm STANAG",
  "numLoaded": 30,
  "capacity": 30,
  "perRoundMass_kg": 0.01231,
  "emptyMass_kg": 0.1207
}
```

A.362 SSTAF/src/framework/mil.sstaf.core/src/test/resources/SoldierFactoryTest/TestKit.json

```
{
  "guns": [
    "M16A1.json"
  ],
  "magazines": [
    "STANAGMagazine.json",
    "STANAGMagazine.json",
    "STANAGMagazine.json",
    "STANAGMagazine.json"
  ],
  "packs": [
    "FannyPack.json"
  ]
}
```

A.363 SSTAF/src/framework/mil.sstaf.core/src/test/resources/SoldierFactoryTest/TestSoldier1.json

```
{
  "class" : "mil.sstaf.core.entity.Soldier",
  "name": "Test Dude",
  "rank": "E6",
  "features": [
  ],
  "configurations": {}
}
```

A.364 SSTAF/src/framework/mil.sstaf.core/src/test/resources/SoldierFactoryTest/TestSoldier2.json

```
{
  "class" : "mil.sstaf.core.entity.Soldier",
  "name": "Test Dude 2",
  "rank": "E6",
  "features": [
  ],
  "configurations": {}
}
```

A.365 SSTAF/src/framework/mil.sstaf.core/src/test/resources/SoldierFactoryTest/TestSoldier6.json

```
{
  "class" : "mil.sstaf.core.entity.Soldier",
  "name": "Test Dude",
  "rank": "E6",
  "features": [],
  "configurations": {}
}
```

A.366 SSTAF/src/framework/mil.sstaf.core/src/test/resources/SoldierFactoryTest/anthro.json

```
{
  "sex": "Female",
  "age": 23,
  "height_cm": 150,
  "span_cm": 150,
  "weight_kg": 50
}
```

A.367 SSTAF/src/framework/mil.sstaf.core/src/test/resources/SoldierFactoryTest/ap.json

```
{
  "serviceName": "Dynamic Aim",
  "majorVersion": 3,
  "minorVersion": 1,
  "requireExact": true,
  "configuration": {}
}
```

A.368 SSTAF/src/framework/mil.sstaf.core/src/test/resources/SoldierFactoryTest/ip.json

```
{
  "serviceName": "ORCA",
  "majorVersion": 3,
  "minorVersion": 1,
  "requireExact": true,
  "configuration": {}
}
```

A.369 SSTAF/src/framework/mil.sstaf.core/src/test/resources/SoldierFactoryTest/kp.json

```
{
  "serviceName": "Santos",
  "majorVersion": 3,
  "minorVersion": 1,
  "requireExact": true,
  "configuration": {}
}
```

A.370 SSTAF/src/framework/mil.sstaf.core/src/test/resources/SoldierFactoryTest/tp.json

```
{
  "serviceName": "ACQUIRE",
  "majorVersion": 3,
  "minorVersion": 1,
  "requireExact": true,
  "configuration": {}
}
```

A.371 SSTAF/src/framework/mil.sstaf.core/src/test/resources/TestAnthroConfig.json

```
{
  "sex": "Female",
  "age": 23,
  "height_cm": 150,
  "span_cm": 150,
  "weight_kg": 50
}
```

A.372 SSTAF/src/framework/mil.sstaf.core/src/test/resources/TestProviderSpec.json

```
{
  "featureName": "Mary Poppins",
  "majorVersion": 3,
  "minorVersion": 1,
  "requireExact": true
}
```

A.373 SSTAF/src/framework/mil.sstaf.core/src/test/resources/UnitFactoryTest/BadUnit1.json

```
{
  "class" : "mil.sstaf.core.entity.Unit",
  "name": "Test Platoon",
  "type": "Platoon",
  "soldiers": [
    {
      "position": "CC",
      "soldier": "../SoldierFactoryTest/TestSoldier1.json"
    },
    {
      "position": "DCC",
      "soldier": "../SoldierFactoryTest/TestSoldier1.json"
    },
    {
      "position": "G2",
      "soldier": "../SoldierFactoryTest/TestSoldier1.json"
    }
  ],
  "subUnits": {},
  "features": [],
  "configurations": {}
}
```

A.374 SSTAF/src/framework/mil.sstaf.core/src/test/resources/UnitFactoryTest/BadUnit2.json

```
{
  "class" : "mil.sstaf.core.entity.Unit",
  "name": "Test Squad",
  "type": "Squad",
  "features": [],
  "configurations": {},
  "soldiers": [
    {
      "soldier": "../SoldierFactoryTest/TestSoldier1.json"
    }
  ],
  "subUnits": [
    {
      "label": "Fire Team Alpha",
      "unit": "FireTeam.json"
    },
    {
      "label": "Fire Team Bravo",
      "unit": "FireTeam.json"
    }
  ]
}
```

A.375 SSTAF/src/framework/mil.sstaf.core/src/test/resources/UnitFactoryTest/BadUnit3.json

```
{
  "class" : "mil.sstaf.core.entity.Unit",
  "name": "Test Squad",
  "type": "Squad",
  "features": [],
  "configurations": {},
  "soldiers": [
    {
      "position": "SL"
    }
  ],
  "subUnits": [
    {
      "label": "Fire Team Alpha"
    },
    {
      "label": "Fire Team Bravo",
      "unit": "FireTeam.json"
    }
  ]
}
```

A.376 SSTAF/src/framework/mil.sstaf.core/src/test/resources/UnitFactoryTest/BadUnit4.json

```
{
  "class" : "mil.sstaf.core.entity.Unit",
  "name": "Test Squad",
  "type": "Squad",
  "features": [],
  "configurations": {},
  "soldiers": [
    {
      "position": "SL",
      "soldier": "../SoldierFactoryTest/TestSoldier1.json"
    }
  ],
  "subUnits": [
    {
      "unit": "FireTeam.json"
    },
    {
      "label": "Fire Team Bravo",
      "unit": "FireTeam.json"
    }
  ]
}
```

A.377 SSTAF/src/framework/mil.sstaf.core/src/test/resources/UnitFactoryTest/BadUnit5.json

```
{
  "class" : "mil.sstaf.core.entity.Unit",
  "name": "Test Squad",
  "type": "Squad",
  "features": [],
  "configurations": {},
  "soldiers": [
    {
      "position": "SL",
      "soldier": "../SoldierFactoryTest/TestSoldier1.json"
    }
  ],
  "subUnits": [
    {
      "label": "Fire Team Alpha"
    },
    {
      "label": "Fire Team Bravo",
      "unit": "FireTeam.json"
    }
  ]
}
```

A.378 SSTAF/src/framework/mil.sstaf.core/src/test/resources/UnitFactoryTest/Company.json

```
{
  "class" : "mil.sstaf.core.entity.Unit",
  "name": "Test Platoon",
  "type": "Platoon",
  "features": [],
  "configurations": {},
  "soldiers": [
    {
      "position": "CC",
      "soldier": "../SoldierFactoryTest/TestSoldier1.json"
    },
    {
      "position": "DCC",
      "soldier": "../SoldierFactoryTest/TestSoldier1.json"
    },
    {
      "position": "G2",
      "soldier": "../SoldierFactoryTest/TestSoldier1.json"
    }
  ],
  "subUnits": [
    {
      "label": "1st Platoon",
      "unit": "Platoon.json"
    },
    {
      "label": "2nd Platoon",
      "unit": "Platoon.json"
    },
    {
      "label": "3rd Platoon",
      "unit": "Platoon.json"
    }
  ]
}
```

A.379 SSTAF/src/framework/mil.sstaf.core/src/test/resources/UnitFactoryTest/FireTeam.json

```
{
  "class" : "mil.sstaf.core.entity.Unit",
  "name": "Test Unit",
  "type": "FireTeam",
  "features": [
  ],
  "configurations": {},
  "soldiers": [
    {
      "position": "TL",
      "soldier": "../SoldierFactoryTest/TestSoldier1.json"
    },
    {
      "position": "R",
      "soldier": "../SoldierFactoryTest/TestSoldier2.json"
    },
    {
      "position": "G",
      "soldier": "../SoldierFactoryTest/TestSoldier1.json"
    },
    {
      "position": "AR",
      "soldier": "../SoldierFactoryTest/TestSoldier2.json"
    }
  ]
}
```

A.380 SSTAF/src/framework/mil.sstaf.core/src/test/resources/UnitFactoryTest/Platoon.json

```
{
  "class" : "mil.sstaf.core.entity.Unit",
  "name": "Test Platoon",
  "type": "Platoon",
  "features": [],
  "configurations": {},
  "soldiers": [
    {
      "position": "PL",
      "soldier": "../SoldierFactoryTest/TestSoldier1.json"
    },
    {
      "position": "DPL",
      "soldier": "../SoldierFactoryTest/TestSoldier1.json"
    }
  ],
  "subUnits": [
    {
      "label": "Squad A",
      "unit": "Squad.json"
    },
    {
      "label": "Squad B",
      "unit": "Squad.json"
    },
    {
      "label": "Squad C",
      "unit": "Squad.json"
    }
  ]
}
```

A.381 SSTAF/src/framework/mil.sstaf.core/src/test/resources/UnitFactoryTest/Squad.json

```
{
  "class" : "mil.sstaf.core.entity.Unit",
  "name": "Test Squad",
  "type": "Squad",
  "features": [
  ],
  "configurations": {},
  "soldiers": [
    {
      "position": "SL",
      "soldier": "../SoldierFactoryTest/TestSoldier1.json"
    }
  ],
  "subUnits": [
    {
      "label": "Fire Team Alpha",
      "unit": "FireTeam.json"
    },
    {
      "label": "Fire Team Bravo",
      "unit": "FireTeam.json"
    }
  ]
}
```

A.382 SSTAF/src/framework/mil.sstaf.core/src/test/resources/aliasTest.json

```
{
  "Alice": 173,
  "Bob": "The String is the Thing",
  "Charley": 3.141592,
  "Della": "Is not a Fella",
  "Edgar": {
    "Stuff": "Stuff"
  },
  "Fanny": [
    1,
    2,
    3,
    4,
    5
  ],
  "Ginny": [
    5,
    4,
    3,
    2,
    1,
    2,
    3,
    4,
    5
  ],
  "Hal": {
    "Best": "Pizza"
  },
  "Iris": 1864,
  "Juliet": 2.71828
}
```

A.383 SSTAF/src/framework/mil.sstaf.core/src/test/resources/arrayTest.json

```
[
  "A",
  "B",
  1,
  2,
  3,
  3.14,
  {
    "C": "Chloe",
    "D": {
      "E": 1701
    }
  }
]
```

A.384 SSTAF/src/framework/mil.sstaf.core/src/test/resources/badFile.json

```
{
  "Alice": "Bob"
  :
  "The String is the Thing",
  "Charley": 3.141592,
  "Della": "Is not a Fella",
  "Edgar": {
    "Stuff": "Stuff"
  },
  "Fanny": [
    1,
    2,
    3,
    4,
    5
  ],
  "Ginny": [
    5,
    4,
    3,
    2,
    1,
    2,
    3,
    4,
    5
  ],
  "Hal": {
    "Best": "Pizza"
  },
  "Iris": 1864,
  "Juliet": 2.71828
}
```

A.385 SSTAF/src/framework/mil.sstaf.core/src/test/resources/emptyConfig.json

```
{
  "class" : "mil.sstaf.core.entity,Unit",
  "modules" : [
  ],
  "modulePaths" : [
  ]
}
```

A.386 SSTAF/src/framework/mil.sstaf.core/src/test/resources/extractionTest/extractMe.txt

```
print "Hello World"
```

A.387 SSTAF/src/framework/mil.sstaf.core/src/test/resources/getObject1.json

```
{
 "fruit" : "banana"
}
```

A.388 SSTAF/src/framework/mil.sstaf.core/src/test/resources/goodProvidersFile1.json

```
{
  "mil.sstaf.core.mocks.PinkyProvider": {
    "serviceName": "Pinky"
  },
  "mil.sstaf.core.mocks.BrainProvider": {
    "serviceName": "Brain"
  }
}
```

A.389 SSTAF/src/framework/mil.sstaf.core/src/test/resources/humanTest1.json

```
{
  "name": "Bob",
  "rank": "LTC",
  "features": [],
  "configurations": {},
  "anthropometry": {
    "weight": "120"
  },
  "injuryProvider": "ORCA",
  "kinematicsProvider": "Simple",
  "aimProvider": "dynamic",
  "targetingProvider": "acquire"
}
```

A.390 SSTAF/src/framework/mil.sstaf.core/src/test/resources/iterateOver1.json

```
{
  "stuff": [
    {
      "name": "A"
    },
    {
      "name": "B"
    },
    {
      "name": "C"
    }
  ]
}
```

A.391 SSTAF/src/framework/mil.sstaf.core/src/test/resources/iterateOver2.json

```
{
  "stuff": [
    {
      "name": "A"
    },
    "AAABBBCCC",
    "DDDEEEFFF",
    {
      "name": "B"
    },
    "GGGHHHIII",
    "JJJKKKLLL",
    {
      "name": "C"
    },
    "MMMNNNOOO"
  ]
}
```

A.392 SSTAF/src/framework/mil.sstaf.core/src/test/resources/jsonTests/badclass.json

```
{
  "class": "mil.sstaf.wrong.json.Message2",
  "ints": [
    1,
    2,
    3,
    4,
    5,
    6
  ]
}
```

A.393 SSTAF/src/framework/mil.sstaf.core/src/test/resources/jsonTests/badref.json

```
{
  "class" : "mil.sstaf.core.json.Message7",
  "msg7" : "./not_there.json"
}
```

A.394 SSTAF/src/framework/mil.sstaf.core/src/test/resources/jsonTests/message2.json

```
{
  "class": "mil.sstaf.core.json.Message2",
  "ints": [
    1,
    2,
    3,
    4,
    5,
    6
  ]
}
```

A.395 SSTAF/src/framework/mil.sstaf.core/src/test/resources/jsonTests/message4.json

```
{
  "class":"mil.sstaf.core.json.Message4",
  "it":"./message2.json"
}
```

A.396 SSTAF/src/framework/mil.sstaf.core/src/test/resources/jsonTests/message5.json

```
{
  "class" : "mil.sstaf.core.json.Message5",
  "msg4" : "./message4.json"
}
```

A.397 SSTAF/src/framework/mil.sstaf.core/src/test/resources/jsonTests/message6.json

```
{
  "class": "mil.sstaf.core.json.Message6",
  "fours": [
    "./message4.json",
    "./message4.json",
    "./message4.json",
    "./message4.json"
  ]
}
```

A.398 SSTAF/src/framework/mil.sstaf.core/src/test/resources/jsonTests/message7.json

```
{
  "class" : "mil.sstaf.core.json.Message7",
  "msg8" : "./message8.json"
}
```

A.399 SSTAF/src/framework/mil.sstaf.core/src/test/resources/jsonTests/message8.json

```
{
  "class" : "mil.sstaf.core.json.Message8",
  "msg7" : "./message7.json"
}
```

A.400 SSTAF/src/framework/mil.sstaf.core/src/test/resources/jsonTests/noClassField.json

```
{
  "ints": [
    1,
    2,
    3,
    4,
    5,
    6
  ]
}
```

A.401 SSTAF/src/framework/mil.sstaf.core/src/test/resources/log4j2-test.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="INFO">
    <Appenders>
        <Console name="Console" target="SYSTEM_OUT">
            <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"/>
        </Console>
    </Appenders>
    <Loggers>
        <Root level="debug">
            <AppenderRef ref="Console"/>
        </Root>
    </Loggers>
</Configuration>
```

A.402 SSTAF/src/framework/mil.sstaf.core/src/test/resources/moduleLayers/SSTAFConfigurationTest1.json

```
{
  "class" : "mil.sstaf.core.configuration.SSTAFConfiguration",

  "moduleLayerDefinition" : {
    "modules": [
      "mil.sstaftest.fred",
      "mil.sstaftest.wilma"
    ],
    "modulePaths": [
      "dir1",
      "dir3"
    ]
  }
}
```

A.403 SSTAF/src/framework/mil.sstaf.core/src/test/resources/moduleLayers/SSTAFConfigurationTest2.json

```
{
  "class": "mil.sstaf.core.configuration.SSTAFConfiguration",
  "moduleLayerDefinition": {
    "modules": [
      "mil.sstaftest.fred",
      "mil.sstaftest.wilma",
      "mil.sstaftest.betty"
    ],
    "modulePaths": [
      "dir1",
      "dir3",
      "dir4"
    ]
  }
}
```

A.404 SSTAF/src/framework/mil.sstaf.core/src/test/resources/moduleLayers/SSTAFConfigurationTest3.json

```
{
  "class": "mil.sstaf.core.configuration.SSTAFConfiguration",
  "moduleLayerDefinition": {
    "modules": [
      "mil.sstaftest.fred",
      "mil.sstaftest.wilma",
      "mil.sstaftest.betty",
      "mil.sstaftest.barney"
    ],
    "modulePaths": [
      "dir1",
      "dir2",
      "dir3",
      "dir4"
    ]
  }
}
```

A.405 SSTAF/src/framework/mil.sstaf.core/src/test/resources/moduleLayers/SSTAFConfigurationTest4.json

```
{
  "class": "mil.sstaf.core.configuration.SSTAFConfigurationToo",
  "moduleLayerDefinition": {
    "modules": [
      "mil.sstaftest.fred",
      "mil.sstaftest.wilma",
      "mil.sstaftest.betty",
      "mil.sstaftest.barney"
    ],
    "modulePaths": [
      "dir1",
      "dir2",
      "dir3",
      "dir4"
    ]
  },
  "more": "This is something more!"
}
```

A.406 SSTAF/src/framework/mil.sstaf.session/build.gradle

```
dependencies {
    implementation project(':framework:mil.sstaf.core')
    implementation project(':features:support:mil.sstaf.blackboard.api')
    implementation group: 'org.slf4j', name: 'slf4j-api', version: '1.7.32'
}
```

A.407 SSTAF/src/framework/mil.sstaf.session/src/main/java/mil/sstaf/session/control/EntityConfiguration.java

```
package mil.sstaf.session.control;

import com.fasterxml.jackson.annotation.JsonTypeInfo;
import com.fasterxml.jackson.databind.annotation.JsonDeserialize;
import lombok.Builder;
import lombok.Getter;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import mil.sstaf.core.entity.BaseEntity;
import mil.sstaf.core.entity.Entity;
import mil.sstaf.core.entity.Force;
import java.util.List;
import java.util.Map;
@SuperBuilder
@Jacksonized
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
public final class EntityConfiguration {
//    public static final String CK_BLUE_FORCE_KEY = "blueForces";
//    public static final String CK_RED_FORCE_KEY = "redForces";
//    public static final String CK_GRAY_FORCE_KEY = "grayForce";
    @Getter
    private final Map<Force, List<BaseEntity>> participants;
}
```

A.408 SSTAF/src/framework/mil.sstaf.session/src/main/java/mil/sstaf/session/control/EntityController.java

```
package mil.sstaf.session.control;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.Builder;
import lombok.Getter;
import lombok.Setter;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import mil.sstaf.blackboard.api.AddEntryRequest;
import mil.sstaf.core.entity.*;
import mil.sstaf.core.features.HandlerContent;
import mil.sstaf.core.json.JsonLoader;
import mil.sstaf.core.util.Injector;
import mil.sstaf.core.util.RNGUtilities;
import mil.sstaf.core.util.SSTAFException;
import mil.sstaf.session.messages.Error;
import mil.sstaf.session.messages.*;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import java.io.File;
import java.nio.file.Path;
import java.util.*;
import java.util.concurrent.*;

/**
 * Central coordinator for processing Events and dispatching messages.
 */
@SuperBuilder
@Jacksonized
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
public final class EntityController extends BaseEntity {
    public static final String SYSTEM_ENTITY_CONTROLLER = "SYSTEM:EntityController";
    private static final Logger logger = LoggerFactory.getLogger(EntityController.class);
    @Getter
    private final Map<Force, List<BaseEntity>> entities;
    @Getter
    private final int executorThreads;
    //
    // Executor
    //
    @Builder.Default
    private Collection<RunAgentsCallable> runAgentsTasks = null;
    @Builder.Default
    private Collection<ProcessEventsCallable> processEventsTasks = null;
    @Builder.Default
    private ExecutorService executorService = null;
    //
    // EntityRegistry
    //
    @Builder.Default
    private EntityRegistry registry = null;
    //
    // ClientProxy - used for interaction with the Session. Enables
    // Entity behavior without exposing Entities through the Session.
    //
    @Builder.Default
    private ClientProxy clientProxy = null;
    @Getter
    @Builder.Default
    private long lastTickTime_ms = 0;
    @Getter
    @Builder.Default
    private long nextEventTime_ms = 0;
```

```
    /**
     * Constructor
     *
     * @param builder the builder that was generated by Lombok
     */
    private EntityController(EntityControllerBuilder<?, ?> builder) {
        super(builder);
        // Replace the user-space id with a system-space id.
        this.executorThreads = builder.executorThreads == 0 ? Runtime.getRuntime().availableProce
ssors() : builder.executorThreads;
        this.id = BlockCounter.systemCounter.getID();
        this.clientProxy = ClientProxy.builder().build();
        this.entities = builder.entities;
        this.handle.setForce(Force.SYSTEM);
        this.runAgentsTasks = new ArrayList<>();
        this.processEventsTasks = new ArrayList<>();
        this.registry = new EntityRegistry();
        this.registry.setClientAddress(clientProxy.getHandle());
        this.registry.registerEntities(entities);
        this.registry.registerEntity(Force.SYSTEM, this);
        this.registry.registerEntity(Force.SYSTEM, clientProxy);
        this.registry.compileEntityMaps();
        this.registry.getSimulationEntities().forEach(entity -> {
            prepareEntity(entity);
            runAgentsTasks.add(new RunAgentsCallable(entity));
            processEventsTasks.add(new ProcessEventsCallable(entity));
        });
        this.clientProxy.setForce(Force.SYSTEM);
        this.clientProxy.setName("ClientProxy");
        this.clientProxy.setRegistry(registry);
        this.clientProxy.init();
        /*
         * Features running in the EntityController probably need access to the registry.
         */
        this.featureManager.injectAll(registry);
        if (this.executorThreads > 1) {
            this.executorService = Executors.newFixedThreadPool(this.executorThreads);
        } else {
            this.executorService = Executors.newSingleThreadExecutor();
        }
        //
        // Create and register a Handler to process messages addressed
        // to the EntityController.
        //
        EntityControllerHandler ech = new EntityControllerHandler(this);
        this.featureManager.register(ech);
        init();
    }
    public static EntityController from(File file) {
        Path p = Path.of(file.getPath());
        JsonLoader jsonLoader = new JsonLoader();
        return jsonLoader.load(p, EntityController.class);
    }
    public void shutdown() {
        this.executorService.shutdown();
    }
    /**
     * Performs final preparations on the Entity
     *
     * @param entity the Entity to prepare
     */
    private void prepareEntity(Entity entity) {
        if (logger.isInfoEnabled()) {
            logger.info("Configuring {}", entity.getName());
        }
        //
        // Need to set the seed for each entity here before init()ing
        //
```

```
        long subSeed = RNGUtilities.generateSubSeed(randomGenerator);
        logger.info("Setting seed in {} to {}", entity.getName(), subSeed);
        Injector.inject(entity, "randomSeed", subSeed);
        entity.injectInFeatures(registry);
        //
        // Initialize it!
        //
        entity.init();
        //
        // Register this controller with each Entity using a message.
        //
        if (entity.canHandle(AddEntryRequest.class)) {
            HandlerContent content = AddEntryRequest.from(handle);
            EntityAction entityAction = EntityAction.builder()
                    .destination(Address.makeExternalAddress(entity.getHandle()))
                    .source(Address.makeExternalAddress(getHandle()))
                    .sequenceNumber(generateSequenceNumber())
                    .respondTo(Address.NOWHERE).content(content).build();
            entity.receive(entityAction);
        }
        //
        // Additional messages?
        //
        //
        // Process messages
        //
        entity.processMessages(Long.MIN_VALUE);
        if (logger.isInfoEnabled()) {
            logger.info("Done configuring {}", entity.getName());
        }
    }
    /**
     * Accepts a {@code BaseSessionCommand} from the client and routes it for processing.
     *
     * @param command the command
     */
    public void submitCommand(final Command command) {
        clientProxy.submitCommand(command);
    }
    /**
     * Accepts a {@code SSTAFEvent} from the client and routes it for processing.
     *
     * @param event the event
     */
    public void submitEvent(final Event event) {
        clientProxy.submitEvent(event);
    }
    /**
     * Gets outbound Messages
     *
     * @return SSTAFResults generated from outbound internal messages
     */
    public List<BaseSessionResult> getMessagesToSession() {
        List<Message> messages = clientProxy.takeInbound();
        List<BaseSessionResult> output = new ArrayList<>();
        messages.forEach(message -> {
            if (message instanceof MessageResponse) {
                MessageResponse mr = (MessageResponse) message;
                BaseSessionResult sstafResult = convertMessageToResult(mr);
                output.add(sstafResult);
            }
        });
        return output;
    }
    /**
     * Returns the depth of the Session Proxy's queue
     *
     * @return the queue depth.
```

```
     */
    public int getSessionProxyQueueDepth() {
        return clientProxy.getQueueDepth();
    }
    /**
     * Processes all the actions and events up to the specified simulation time.
     * <p>
     * This is the central method in the Soldier and Squad Trade-space Analysis Framework.
     *
     * @param currentTime_ms the current simulation time.
     * @return the time of the next added event
     */
    public SessionTickResult tick(long currentTime_ms) {
        logger.debug("Executing tick at {}", currentTime_ms);
        List<Future<Long>> nextTimes1;
        lastTickTime_ms = currentTime_ms;
        try {
            runAgentsTasks.forEach(task -> task.setCurrentTime(currentTime_ms));
            nextTimes1 = executorService.invokeAll(runAgentsTasks);
        } catch (InterruptedException e) {
            e.printStackTrace();
            nextTimes1 = List.of();
        }
        routeMessages();
        //
        // Entity controller runs in reverse. First messages are received and processed,
        // then Agents wrap up global tasks and push outcomes and global state
        //
        this.processMessages(currentTime_ms);
        this.runAgents(currentTime_ms);
        routeMessages();
        List<Future<Long>> nextTimes2;
        try {
            processEventsTasks.forEach(task -> task.setCurrentTime_ms(currentTime_ms));
            nextTimes2 = executorService.invokeAll(processEventsTasks);
        } catch (InterruptedException e) {
            e.printStackTrace();
            nextTimes2 = List.of();
        }
        routeMessages();
        List<BaseSessionResult> toSession = getMessagesToSession();
        long nextEventTime_ms = Long.min(getMinTime(nextTimes1), getMinTime(nextTimes2));
        return SessionTickResult.builder().nextEventTime_ms(nextEventTime_ms)
                .messagesToClient(toSession)
                .build();
    }
    /**
     * Ticks the simulation again using the last tick time.
     * <p>
     * Commands and queries are only executed when a tick happens. Consequently,
     * to make a query or issue a command without changing the state of the
     * simulation the approach is to invoke tick again at the same time. Features
     * should note that the current time is the same and that no state-changing
     * actions should be taken.
     * <p>
     * TODO: A Feature compliance test needs to be developed that ensures that commands and queri
es are idempotent if the time is unchanged.
     *
     * @return a {@code SessionTickResult} containing any results from the tick.
     */
    public SessionTickResult tickAgain() {
        return tick(lastTickTime_ms);
    }

    /**
     * Determines the time of the next event after all events have been processed.
     *
     * @param times the futures for the processing tasks
```

```
     * @return the minimum time
     */
    private long getMinTime(List<Future<Long>> times) {
        long minTime_ms = Long.MAX_VALUE;
        logger.info("times = {}", times);
        for (Future<Long> fd : times) {
            logger.info("fd = {}", fd);
            try {
                long nt = fd.get();
                minTime_ms = Math.min(minTime_ms, nt);
            } catch (InterruptedException e) {
                logger.error("Interrupted!");
                e.printStackTrace();
            } catch (ExecutionException e) {
                logger.error("Broken! " + e.getMessage());
                e.printStackTrace();
            }
        }
        return minTime_ms;
    }
    /**
     * Resolves a path to find the EntityHandle
     *
     * @param path the entity path to look up
     * @return an {@code Optional} that contains the found {@code EntityHandle} or
     * is empty if the path was invalid.
     */
    public Optional<EntityHandle> getHandleFromPath(String path) {
        return registry.getHandle(path);
    }
    /**
     * Takes messages from a MessageDriven object and routes them to the specified
     * recipients.
     *
     * @param routeFrom the MessageDriven instance from which to task the messages.
     */
    private void routeFromMessageDriven(final Entity routeFrom) {
        List<Message> messages = routeFrom.takeOutbound();
        logger.info("Routing messages from {}, got {} messages to route", routeFrom.getName(), me
ssages.size());
        messages.forEach(message -> {
            if (message == null) {
                logger.warn("Message is null");
            } else if (message.getDestination() == null) {
                logger.warn("Message destination is null, source = {}, content = {}", message.get
Source(), message.getContent());
            } else if (message.getDestination().equals(Address.NOWHERE)) {
                logger.debug("Dropping message from {} to NOWHERE, contents = {}", message.getSou
rce(), message.getContent().getClass());
            } else {
                logger.debug("Routing from {} to {}, contents = {}", message.getSource(), message
.getDestination().entityHandle.getName(), message.getContent().getClass());
                Optional<Entity> optionalEntity = registry.getEntityByHandle(message.getDestinati
on().entityHandle);
                optionalEntity.ifPresent(entity -> entity.receive(message));
            }
        });
    }
    /**
     * Routes all messages from all entities
     */
    private void routeMessages() {
        logger.info("Routing messages");
        var allEntities = registry.getAllEntities();
        // split for debugging
        allEntities.forEach(this::routeFromMessageDriven);
    }
    public BaseSessionResult convertMessageToResult(final MessageResponse response) {
```

```
        if (response instanceof ErrorResponse) {
            ErrorResponse errorResponse = (ErrorResponse) response;
            return convertError(errorResponse);
        } else {
            return convertSuccess(response);
        }
    }
    private Error convertError(final ErrorResponse errorResponse) {
        return Error.builder()
                .id(errorResponse.getMessageID())
                .entityPath(errorResponse.getSource().entityHandle.getForcePath())
                .throwable(errorResponse.getThrowable())
                .build();
    }
    private BaseSessionResult convertSuccess(final MessageResponse response) {
        return CommandResult.builder()
                .id(response.getMessageID())
                .entityPath(response.getSource().entityHandle.getForcePath())
                .content(response.getContent())
                .build();
    }
    public SortedSet<EntityHandle> getSimulationEntityHandles() {
        SortedSet<EntityHandle> handles = new TreeSet<>();
        registry.getSimulationEntities().forEach(entity -> handles.add(entity.getHandle()));
        return handles;
    }
    public List<String> getEntityPaths() {
        List<String> paths = new ArrayList<>();
        for (var entry : entities.entrySet()) {
            Force f = entry.getKey();
            for (var entity : entry.getValue()) {
                String fullPath = f.name() + ENTITY_PATH_DELIMITER + entity.getPath();
                paths.add(fullPath);
            }
        }
        return paths;
    }
    public String getPath() {
        return SYSTEM_ENTITY_CONTROLLER;
    }
    @SuperBuilder
    static class ClientProxy extends BaseEntity {
        @Setter
        private EntityRegistry registry;
        /**
         * Constructor
         *
         * @param builder the {@code Builder} to use to construct the {@code Entity}
         */
        protected ClientProxy(ClientProxyBuilder<?, ?> builder) {
            super(builder);
            handle.setForce(Force.SYSTEM);
        }
        @Override
        public String getPath() {
            return "SYSTEM:ClientProxy";
        }

        /**
         * Updates the command with the EntityHandle if it isn't set
         *
         * @param command the {@code BaseSessionCommand} to update.
         */
        private void resolvePath(Command command) {
            Optional<EntityHandle> optEH = registry.getHandle(command.getRecipientPath());
            optEH.ifPresentOrElse(command::setHandle,
                    () -> {
```

```
                        throw new SSTAFException("Entity Path '" + command.getRecipientPath() + "
' does not exist");
                    }
            );
        }
        private void submitCommand(final Command command) {
            resolvePath(command);
            var b = EntityAction.builder()
                    .destination(Address.makeExternalAddress(command.getHandle()))
                    .source(Address.makeExternalAddress(getHandle()))
                    .content(command.getContent())
                    .respondTo(Address.makeExternalAddress(getHandle()));
            outboundQueue.offer(b.build());
        }
        public void submitEvent(final Event event) {
            resolvePath(event);
            var b = EntityEvent.builder().destination(Address.makeExternalAddress(event.getHandle
())).source(Address.makeExternalAddress(getHandle())).eventTime_ms(event.getEventTime_ms()).conte
nt(event.getContent()).respondTo(Address.makeExternalAddress(getHandle()));
            outboundQueue.offer(b.build());
        }
        public int getQueueDepth() {
            return outboundQueue.size();
        }
    }
    /**
     *
     */
    private static class RunAgentsCallable implements Callable<Long> {
        final Entity entity;
        long currentTime_ms;
        RunAgentsCallable(Entity entity) {
            this.entity = entity;
        }
        void setCurrentTime(final long currentTime_ms) {
            logger.info("Updating time in RunAgentsCallable to {}", currentTime_ms);
            this.currentTime_ms = currentTime_ms;
        }
        public Long call() {
            logger.info("Invoking runAgents for {}", entity.getName());
            long res = entity.runAgents(currentTime_ms);
            logger.info("runAgents for {} returned {}", entity.getName(), res);
            return res;
        }
    }
    private static class ProcessEventsCallable implements Callable<Long> {
        final Entity entity;
        long currentTime_ms;
        ProcessEventsCallable(Entity entity) {
            this.entity = entity;
        }
        void setCurrentTime_ms(final long currentTime_ms) {
            logger.info("Updating time in ProcessEventsCallable to {}", currentTime_ms);
            this.currentTime_ms = currentTime_ms;
        }
        public Long call() {
            logger.info("Invoking processMessages for {}", entity.getName());
            long res = entity.processMessages(currentTime_ms);
            logger.debug("processMessages for {} returned {}", entity.getName(), res);
            return res;
        }
    }
}
```

A.409 SSTAF/src/framework/mil.sstaf.session/src/main/java/mil/sstaf/session/control/EntityControllerHandler.java

```
package mil.sstaf.session.control;
import mil.sstaf.core.entity.Address;
import mil.sstaf.core.entity.Message;
import mil.sstaf.core.features.BaseHandler;
import mil.sstaf.core.features.HandlerContent;
import mil.sstaf.core.features.ProcessingResult;
import mil.sstaf.session.messages.GetEntitiesQuery;
import mil.sstaf.session.messages.GetEntitiesResponse;
import java.util.Objects;
public class EntityControllerHandler extends BaseHandler {

    private final EntityController entityController;
    protected EntityControllerHandler(EntityController ec) {
        super("Entity Controller", 1, 0, 0, false, "Provides access to the EntityController via m
essages");
        entityController = Objects.requireNonNull(ec, "ec");
    }
    @Override
    public ProcessingResult process(HandlerContent arg, long scheduledTime_ms, long currentTime_m
s, Address from, long id, Address respondTo) {
        if (arg instanceof GetEntitiesQuery) {
            var x = entityController.getEntities();
            var c = GetEntitiesResponse.builder().entityMap(x).build();
            Message out = buildNormalResponse(c, id, respondTo);
            return ProcessingResult.of(out);
        } else {
            return ProcessingResult.empty();
        }
    }
}
```

A.410 SSTAF/src/framework/mil.sstaf.session/src/main/java/mil/sstaf/session/control/Session.java

```
package mil.sstaf.session.control;
import mil.sstaf.core.entity.EntityHandle;
import mil.sstaf.session.messages.BaseSessionCommand;
import mil.sstaf.session.messages.Command;
import mil.sstaf.session.messages.Event;
import mil.sstaf.session.messages.SessionTickResult;
import java.util.Objects;
import java.util.SortedSet;
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;
/**
 * Client interface to the Soldier and Squad Trade-space Analysis Framework
 * <p>
 * The Session class encapsulates the entire SSTAF system behind a simple message-based interface
. The
 * intent of Session is to enable the capability of SSTAF to be enhanced without client applicati
ons
 * needing to change. It is expected that the Session class will remain very stable.
 */
public final class Session implements AutoCloseable {
    private final boolean asynch;
    private final EntityController entityController;
    private final ExecutorService executorService;
    private long currentTime_ms;
    private Session(SessionConfiguration sessionConfiguration, EntityController entityController)
{
        Objects.requireNonNull(sessionConfiguration, "sessionConfiguration");
        Objects.requireNonNull(entityController, "entityController");
        this.entityController = entityController;
        this.asynch = sessionConfiguration.isAsync();
        this.executorService = this.asynch ? Executors.newSingleThreadExecutor() : null;
    }
    public static Session of(SessionConfiguration sessionConfiguration, EntityController entityCo
nfig) {
        return new Session(sessionConfiguration, entityConfig);
    }
    /**
     * Submits a BaseSessionCommand or SSTAFEvent to the environment.
     * <p>
     * When submitted, the BaseSessionCommand or SSTAFEvent is routed to the Entity specified
     * by the EntityHandle within the commend.
     *
     * @param command The command to process.
     */
    public void submit(final BaseSessionCommand command) {
        if (command instanceof Event) {
            Event event = (Event) command;
            entityController.submitEvent(event);
        } else {
            entityController.submitCommand((Command) command);
        }
    }
    /**
     * {@inheritDoc}
     */
    public void close() {
        if (entityController != null) {
            entityController.shutdown();
        }
        if (executorService != null) {
```

```
            this.executorService.shutdown();
        }
    }
    /**
     * Advances the simulation to the specified time and processes all pending commands
     * and events.
     * <p>
     * Tick() advances the SSTAF simulation clock and triggers all Entities within the
     * environment to tick forward to the same time. When each Entity ticks, it wll
     * process all enqueued commands and any events that are scheduled up to the
     * provided currentTime value. The results for all of the Entities are presented back
     * to the caller as a List of SSTAFResults.
     *
     * @param currentTime_ms the new current simulation time.
     * @return a {@code SessionTickResult} that contains the next event time and outbound message
s.
     */
    public SessionTickResult tick(final long currentTime_ms) {
        if (asynch) {
            throw new IllegalStateException("Session was configured for asynchronous use");
        }
        return entityController.tick(currentTime_ms);
    }
    /**
     * Executes a tick without advancing the simulation clock.
     * <p>
     * TickAgain() forces all {@code Entity} instances to process pending
     * messages using the last simulation time. This method is primarily
     * intended to process state queries.
     *
     * @return a {@code SessionTickResult} that contains the next event time and outbound message
s.
     */
    public SessionTickResult tickAgain() {
        if (asynch) {
            throw new IllegalStateException("Session was configured for asynchronous use");
        }
        return entityController.tickAgain();
    }

    /**
     * Processes a tick asynchronously using an ExecutorService.
     *
     * @param currentTime_ms the new current simulation time.
     * @return A Future for the results of the tick();
     */
    @SuppressWarnings("unused")
    public Future<SessionTickResult> asyncTick(final long currentTime_ms) {
        if (!asynch) {
            throw new IllegalStateException("Attempted to use asyncTick() when Session was not co
nfigured for asynchronous use");
        }
        return executorService.submit(new TickCallable(currentTime_ms));
    }
    /**
     * Produces a {@code SessionTickResult} that includes the current simulation time
     * but no {@code BaseSessionResult}s.
     *
     * @return an empty {@code SessionTickResult}
     */
    public SessionTickResult getEmptyTickResult() {
        return SessionTickResult.builder().nextEventTime_ms(currentTime_ms).build();
    }

    /**
     * Provides a SortedSet of EntityHandles for all of the Entities in
     * the environment
     *
```

```
     * @return a SortedSet of EntityHandles
     */
    public SortedSet<EntityHandle> getEntities() {
        return entityController.getSimulationEntityHandles();
    }
    public EntityController getEntityController() {
        return entityController;
    }
    /**
     * Callable for executing a tick (duh!)
     */
    private class TickCallable implements Callable<SessionTickResult> {
        public TickCallable(long currentTime_ms) {
            Session.this.currentTime_ms = currentTime_ms;
        }
        @Override
        public SessionTickResult call() {
            return tick(currentTime_ms);
        }

    }
}
```

A.411 SSTAF/src/framework/mil.sstaf.session/src/main/java/mil/sstaf/session/control/SessionConfiguration.java

```
package mil.sstaf.session.control;

import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.Getter;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import mil.sstaf.core.configuration.SSTAFConfiguration;
import java.util.List;
@SuperBuilder
@Jacksonized
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
public class SessionConfiguration extends SSTAFConfiguration {
    @Getter
    private final boolean async;
}
```

A.412 SSTAF/src/framework/mil.sstaf.session/src/main/java/mil/sstaf/session/messages/BaseSessionCommand.java

```
package mil.sstaf.session.messages;
import com.fasterxml.jackson.annotation.JsonIgnore;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import java.util.concurrent.atomic.AtomicLong;
import static lombok.Builder.Default;
@Jacksonized
@SuperBuilder
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
@EqualsAndHashCode
public class BaseSessionCommand {
    private static final AtomicLong counter = new AtomicLong(0);
    @Getter
    @Default
    @JsonIgnore
    private long id = counter.getAndIncrement();
}
```

A.413 SSTAF/src/framework/mil.sstaf.session/src/main/java/mil/sstaf/session/messages/BaseSessionResult.java

```
package mil.sstaf.session.messages;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.Setter;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
@Jacksonized
@SuperBuilder
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
@EqualsAndHashCode
public class BaseSessionResult {
    @Getter
    @Setter
    private long id;
    public BaseSessionResult(final long id) {
        this.id = id;
    }
}
```

A.414 SSTAF/src/framework/mil.sstaf.session/src/main/java/mil/sstaf/session/messages/Command.java

```
package mil.sstaf.session.messages;
import com.fasterxml.jackson.annotation.JsonIgnore;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.*;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import mil.sstaf.core.entity.EntityHandle;
import mil.sstaf.core.features.HandlerContent;
import mil.sstaf.session.control.Session;
@Jacksonized
@SuperBuilder
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
@EqualsAndHashCode(callSuper = true)
public class Command extends BaseSessionCommand {
    @Getter
    @NonNull
    private final String recipientPath;
    @Getter
    @NonNull
    private final HandlerContent content;
    @Getter
    @Setter
    @Builder.Default
    @JsonIgnore
    private EntityHandle handle = null;
    /**
     * Queries the provided {@code Session} to determine and assign the
     * {@code EntityHandle}.
     *
     * @param session The {@code Session}
     */
    public void setHandleFromSession(Session session) {
        session.getEntityController()
                .getHandleFromPath(recipientPath)
                .ifPresent(this::setHandle);
    }
}
```

A.415 SSTAF/src/framework/mil.sstaf.session/src/main/java/mil/sstaf/session/messages/CommandResult.java

```
package mil.sstaf.session.messages;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.ToString;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import mil.sstaf.core.features.HandlerContent;
@Jacksonized
@SuperBuilder
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
@EqualsAndHashCode(callSuper = true)
@ToString
public class CommandResult extends BaseSessionResult {
    @Getter
    private final String entityPath;
    @Getter
    private final HandlerContent content;
}
```

A.416 SSTAF/src/framework/mil.sstaf.session/src/main/java/mil/sstaf/session/messages/Error.java

```
package mil.sstaf.session.messages;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
@Jacksonized
@SuperBuilder
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
@EqualsAndHashCode(callSuper = true)
public class Error extends BaseSessionResult {
    @Getter
    private final Throwable throwable;
    @Getter
    private final String entityPath;
}
```

A.417 SSTAF/src/framework/mil.sstaf.session/src/main/java/mil/sstaf/session/messages/Event.java

```
package mil.sstaf.session.messages;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import mil.sstaf.core.entity.EntityHandle;
import mil.sstaf.core.features.HandlerContent;
@Jacksonized
@SuperBuilder
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
@EqualsAndHashCode(callSuper = true)
public class Event extends Command {
    @Getter
    private final long eventTime_ms;
}
```

A.418 SSTAF/src/framework/mil.sstaf.session/src/main/java/mil/sstaf/session/messages/GetEntitiesQuery.java

```
package mil.sstaf.session.messages;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.EqualsAndHashCode;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import mil.sstaf.core.features.HandlerContent;
@SuperBuilder
@Jacksonized
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
@EqualsAndHashCode(callSuper = true)
public class GetEntitiesQuery extends HandlerContent {
}
```

A.419 SSTAF/src/framework/mil.sstaf.session/src/main/java/mil/sstaf/session/messages/GetEntitiesResponse.java

```
package mil.sstaf.session.messages;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import mil.sstaf.core.entity.BaseEntity;
import mil.sstaf.core.entity.Force;
import mil.sstaf.core.features.HandlerContent;
import java.util.List;
import java.util.Map;
@SuperBuilder
@Jacksonized
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
@EqualsAndHashCode(callSuper = true)
public class GetEntitiesResponse extends HandlerContent {
    @Getter
    private final Map<Force, List<BaseEntity>> entityMap;
}
```

A.420 SSTAF/src/framework/mil.sstaf.session/src/main/java/mil/sstaf/session/messages/SessionTickResult.java

```
package mil.sstaf.session.messages;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import java.util.List;
@SuperBuilder
@Jacksonized
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
@EqualsAndHashCode(callSuper = true)
public class SessionTickResult extends BaseSessionResult {
    @Getter
    private final long nextEventTime_ms;
    @Getter
    private final List<BaseSessionResult> messagesToClient;
}
```

A.421 SSTAF/src/framework/mil.sstaf.session/src/main/java/module-info.java

```
module mil.sstaf.session {
    exports mil.sstaf.session.messages;
    exports mil.sstaf.session.control;
    requires transitive mil.sstaf.core;
    requires org.slf4j;

    requires mil.sstaf.blackboard.api;
    opens mil.sstaf.session.control to com.fasterxml.jackson.databind;
    opens mil.sstaf.session.messages to com.fasterxml.jackson.databind;
}
```

A.422 SSTAF/src/framework/mil.sstaf.session/src/test/java/mil/sstaf/session/SessionFactoryTest.java

```
package mil.sstaf.session;
import mil.sstaf.core.json.JsonLoader;
import mil.sstaf.session.control.Session;
import mil.sstaf.session.control.SessionConfiguration;
import mil.sstaf.session.control.EntityController;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;
import java.nio.file.Path;
class SessionFactoryTest {
    @Test
    void canReadGoodSession() {
        Assertions.assertDoesNotThrow(() -> {
                    Path sessionPath = Path.of("src/test/resources/SessionFactoryTest/session1.js
on");
                    Path entityPath = Path.of("src/test/resources/EntityControllerFactoryTest/Ent
ityConfig1.json");
                    JsonLoader loader = new JsonLoader();
                    SessionConfiguration sessionConfiguration = loader.load(sessionPath, SessionC
onfiguration.class);
                    EntityController entityController = EntityController.from(entityPath.toFile()
);
                    Session session = Session.of(sessionConfiguration, entityController);
                }
        );
    }
}
```

A.423 SSTAF/src/framework/mil.sstaf.session/src/test/java/mil/sstaf/session/SessionTest.java

```
package mil.sstaf.session;
import mil.sstaf.core.entity.*;
import mil.sstaf.session.control.EntityController;
import mil.sstaf.session.control.Session;
import mil.sstaf.session.control.SessionConfiguration;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import java.util.List;
import java.util.Map;
import static org.junit.jupiter.api.Assertions.*;
class SessionTest {
    private EntityController entityController;
    @BeforeEach
    void setup() {
        Unit bob = Unit.builder().name("Bob").build();
        Map<Force, List<BaseEntity>> map = Map.of(Force.BLUE, List.of(bob));
        entityController = EntityController.builder().entities(map).build();
        assertNotNull(entityController);
        var ub = Unit.builder();
    }
    @Test
    void newBuilder() {
        Assertions.assertDoesNotThrow(() -> {
            SessionConfiguration config = SessionConfiguration.builder().build();
            Session session = Session.of(config, entityController);
        });
    }
//    @Test
//    void submit() {
//        Session.ofBuilder bldr = Session.builder();
//        bldr.withController(entityController);
//        Session session = bldr.build();
//
//        Collection<EntityHandle> allPaths = entityController.getSimulationEntityHandles();
//        BaseSessionCommand cmd = new BaseSessionCommand(allPaths.iterator().next(), "This is a
test");
//        session.submit(cmd);
//        assertEquals(1, entityController.getSessionProxyQueueDepth());
//    }
//
//    @Test
//    void tick() {
//        Session.Builder bldr = Session.builder();
//        bldr.withController(entityController);
//        Session session = bldr.build();
//
//        Collection<EntityHandle> allPaths = entityController.getSimulationEntityHandles();
//        BaseSessionCommand cmd = new BaseSessionCommand(allPaths.iterator().next(), "This is a
test");
//        session.submit(cmd);
//
//        SessionTickResult tickResult = session.tick(10000);
//
//        List<BaseSessionResult> results = tickResult.getMessagesToClient();
//        assertEquals(1, results.size());
//        //
//        // The result should be an error because there are no handlers
//        // registered in the EntityController
//        //
//        Object x = results.get(0);
//        assertTrue(results.get(0) instanceof Error);
```

```
//        BaseSessionResult res = results.get(0);
//        if (res instanceof Error) {
//            Error error = (Error) res;
//            assertNull(error.getContent());
//            assertNotNull(error.getThrowable());
//        }
//    }
//
//    @Test
//    void getEntities() {
//        Session.Builder bldr = Session.builder();
//        bldr.withController(entityController);
//        Session session = bldr.build();
//        assertEquals(entityController.getSimulationEntityHandles().size(), session.getEntities(
).size());
//    }
}
```

A.424 SSTAF/src/framework/mil.sstaf.session/src/test/java/mil/sstaf/session/control/EntityControllerFactoryTest.java

```
package mil.sstaf.session.control;
import mil.sstaf.core.entity.MessageDriven;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.function.Executable;
class EntityControllerFactoryTest {
}
```

A.425 SSTAF/src/framework/mil.sstaf.session/src/test/java/mil/sstaf/session/control/EntityControllerTest.java

```
package mil.sstaf.session.control;
import mil.sstaf.core.entity.*;
import mil.sstaf.core.features.ExceptionCommand;
import mil.sstaf.core.features.StringContent;
import mil.sstaf.session.messages.Error;
import mil.sstaf.session.messages.*;
import org.junit.jupiter.api.*;
import java.io.File;
import java.util.Collection;
import java.util.List;
import java.util.Map;
import static org.junit.jupiter.api.Assertions.*;
class EntityControllerTest {
    private EntityController entityController;
    private Unit unit;
    @BeforeEach
    void setup() {
        var ub = Unit.builder();
        ub.name("Bob");
        unit = ub.build();
        unit.init();
        var ecb = EntityController.builder();
        assertNotNull(ecb);
        ecb.entities(Map.of(Force.BLUE, List.of(unit)));
        entityController = ecb.build();
    }

    @Nested
    @DisplayName("Test basic scenarios")
    class BasicTests {
        @Test
        @DisplayName("Confirm that the EntityController is built successfully")
        void testControllerBuilder() {
            assertNotNull(entityController);
        }
        @Test
        @DisplayName("Confirm that a BaseSessionCommand can be submitted and the queue holds it")
        void testSubmitCommand() {
            Collection<EntityHandle> allPaths = entityController.getSimulationEntityHandles();
            assertEquals(1, allPaths.size());
            Command cmd = Command.builder().
                    recipientPath(allPaths.iterator().next().getPath())
                    .content(StringContent.builder().message("This is a test").build())
                    .build();
            entityController.submitCommand(cmd);
            assertEquals(1, entityController.getSessionProxyQueueDepth());
        }
        @Test
        @DisplayName("Confirm that a BaseSessionCommand can be submitted and the queue holds it")
        void testSubmitEvent() {
            Collection<EntityHandle> allPaths = entityController.getSimulationEntityHandles();
            Event event = Event.builder()
                    .recipientPath(allPaths.iterator().next().getPath())
                    .content(StringContent.builder().message("This is a test").build())
                    .eventTime_ms(10000)
                    .build();
            entityController.submitEvent(event);
            assertEquals(1, entityController.getSessionProxyQueueDepth());
        }
        @Test
        @DisplayName("Confirm that getSimulationEntityHandles() works as expected")
        void getAllEntityHandles() {
```

```
            Collection<EntityHandle> allPaths = entityController.getSimulationEntityHandles();
            assertEquals(1, allPaths.size());
        }
        @Test
        @DisplayName("Confirm that getLastTimeTime_ms works")
        void testTick() {
            Collection<EntityHandle> allPaths = entityController.getSimulationEntityHandles();
            Command cmd = Command.builder()
                    .recipientPath(allPaths.iterator().next().getForcePath())
                    .content(StringContent.builder().message("This is a test").build())
                    .build();
            entityController.submitCommand(cmd);
            assertEquals(1, entityController.getSessionProxyQueueDepth());
            int currentTime_ms = 10000;
            SessionTickResult tickResult = entityController.tick(currentTime_ms);
            assertNotNull(tickResult);
            assertEquals(currentTime_ms, entityController.getLastTickTime_ms());
        }
    }
    @Nested
    @DisplayName("Test failure modes")
    class FailureTests {
        @Test
        @DisplayName("Confirm that dispatching a message to an entity without an appropriate hand
le produces and error message")
        void testTick() {
            Collection<EntityHandle> allPaths = entityController.getSimulationEntityHandles();
            Command cmd = Command.builder()
                    .recipientPath(allPaths.iterator().next().getForcePath())
                    .content( StringContent.builder().message("This is a test").build())
                    .build();
            entityController.submitCommand(cmd);
            assertEquals(1, entityController.getSessionProxyQueueDepth());
            //
            // Spoon!
            //
            SessionTickResult tickResult = entityController.tick(10000);
            assertEquals(1, tickResult.getMessagesToClient().size());
            BaseSessionResult out = tickResult.getMessagesToClient().get(0);
            //
            // The result should be an error because there are no handlers
            // registered in Bob.
            //
            assertTrue(out instanceof Error);
        }
        @Test
        @DisplayName("Confirm that an error message can be converted to a result.")
        void testConvertErrorMessageToResult() {
            var b = ErrorResponse.builder()
                    .sequenceNumber(1234)
                    .messageID(3)
                    .errorDescription("It's broken")
                    .destination(Address.makeExternalAddress(unit.getHandle()))
                    .source(Address.makeExternalAddress(unit.getHandle()))
                    .content(ExceptionCommand.builder().thrown(new Throwable()).build());
            ErrorResponse er = b.build();
            BaseSessionResult sr = entityController.convertMessageToResult(er);
            assertNotNull(sr);
            assertTrue(sr instanceof Error);
            assertEquals("BLUE" + Entity.ENTITY_PATH_DELIMITER + "Bob", ((Error)sr).getEntityPath
());
        }
    }

    @Nested
    @DisplayName("EntityController.Factory tests")
    class ECTTests {
        @Test
```

```
        void testSimpleParseAndBuild() {
            File file = new File("src/test/resources/EntityControllerFactoryTest/EntityConfig1.js
on");
            EntityController entityController = EntityController.from(file);
            Assertions.assertNotNull(entityController);
            Assertions.assertTrue(entityController.getId() >= MessageDriven.BlockCounter.SYSTEM_B
LOCK_BEGIN);
            Assertions.assertTrue(entityController.getId() < MessageDriven.BlockCounter.USER_BLOC
K_BEGIN);
        }
        @Test
        void testParseAndBuild1FireTeam() {
            File file = new File("src/test/resources/EntityControllerFactoryTest/EntityConfig1Fir
eTeam.json");
            EntityController entityController = EntityController.from(file);
            Assertions.assertNotNull(entityController);
            Assertions.assertTrue(entityController.getId() >= MessageDriven.BlockCounter.SYSTEM_B
LOCK_BEGIN);
            Assertions.assertTrue(entityController.getId() < MessageDriven.BlockCounter.USER_BLOC
K_BEGIN);
            Assertions.assertEquals(5, entityController.getSimulationEntityHandles().size());
        }
        @Test
        void testParseAndBuild1Squad() {
            File file = new File("src/test/resources/EntityControllerFactoryTest/EntityConfig1Squ
ad.json");
            EntityController entityController = EntityController.from(file);
            Assertions.assertNotNull(entityController);
            Assertions.assertTrue(entityController.getId() >= MessageDriven.BlockCounter.SYSTEM_B
LOCK_BEGIN);
            Assertions.assertTrue(entityController.getId() < MessageDriven.BlockCounter.USER_BLOC
K_BEGIN);
            Assertions.assertEquals(12, entityController.getSimulationEntityHandles().size());
        }
        @Test
        void testParseAndBuild1Platoon() {
            File file = new File(
                    "src/test/resources/EntityControllerFactoryTest/EntityConfig1Platoon.json");
            EntityController entityController = EntityController.from(file);
            Assertions.assertNotNull(entityController);
            Assertions.assertTrue(entityController.getId() >= MessageDriven.BlockCounter.SYSTEM_B
LOCK_BEGIN);
            Assertions.assertTrue(entityController.getId() < MessageDriven.BlockCounter.USER_BLOC
K_BEGIN);
            Assertions.assertEquals(39, entityController.getSimulationEntityHandles().size());
        }
        @Test
        void testParseAndBuild10Platoons() {
            File file = new File(
                    "src/test/resources/EntityControllerFactoryTest/EntityConfig10Platoons.json")
;
            EntityController entityController = EntityController.from(file);
            Assertions.assertNotNull(entityController);
            Assertions.assertTrue(entityController.getId() >= MessageDriven.BlockCounter.SYSTEM_B
LOCK_BEGIN);
            Assertions.assertTrue(entityController.getId() < MessageDriven.BlockCounter.USER_BLOC
K_BEGIN);
            Assertions.assertEquals(390, entityController.getSimulationEntityHandles().size());
        }
    }
}
```

A.426 SSTAF/src/framework/mil.sstaf.session/src/test/resources/EntityControllerFactoryTest/EntityConfig1.json

```
{
  "class": "mil.sstaf.session.control.EntityController",
  "entities": {
    "BLUE": [],
    "RED": [],
    "GRAY": []
  },
  "features": [],
  "configurations": {
  },
  "randomSeed": 3
}
```

A.427 SSTAF/src/framework/mil.sstaf.session/src/test/resources/EntityControllerFactoryTest/EntityConfig10Platoons.json

```
{
  "class": "mil.sstaf.session.control.EntityController",
  "entities": {
    "BLUE": [
      "../UnitConfiguration/Platoon.json",
      "../UnitConfiguration/Platoon.json",
      "../UnitConfiguration/Platoon.json",
      "../UnitConfiguration/Platoon.json",
      "../UnitConfiguration/Platoon.json"
    ],
    "RED": [
      "../UnitConfiguration/Platoon.json",
      "../UnitConfiguration/Platoon.json",
      "../UnitConfiguration/Platoon.json",
      "../UnitConfiguration/Platoon.json",
      "../UnitConfiguration/Platoon.json"
    ],
    "GRAY": []
  },
  "features": [],
  "configurations": {},
  "randomSeed": 3
}
```

A.428 SSTAF/src/framework/mil.sstaf.session/src/test/resources/EntityControllerFactoryTest/EntityConfig1FireTeam.json

```
{
  "class": "mil.sstaf.session.control.EntityController",
  "entities": {
    "BLUE": [
      "../UnitConfiguration/FireTeam.json"
    ],
    "RED": [],
    "GRAY": []
  },
  "features": [],
  "configurations": {},
  "randomSeed": 3
}
```

A.429 SSTAF/src/framework/mil.sstaf.session/src/test/resources/EntityControllerFactoryTest/EntityConfig1Platoon.json

```
{
  "class": "mil.sstaf.session.control.EntityController",
  "entities": {
    "BLUE": [
      "../UnitConfiguration/Platoon.json"
    ],
    "RED": [],
    "GRAY": []
  },
  "features": [],
  "configurations": {},
  "randomSeed": 3
}
```

A.430 SSTAF/src/framework/mil.sstaf.session/src/test/resources/EntityControllerFactoryTest/EntityConfig1Squad.json

```
{
  "class": "mil.sstaf.session.control.EntityController",
  "entities": {
    "BLUE": [
      "../UnitConfiguration/Squad.json"
    ],
    "RED": [],
    "GRAY": []
  },
  "features": [],
  "configurations": {},
  "randomSeed": 3
}
```

A.431 SSTAF/src/framework/mil.sstaf.session/src/test/resources/SessionFactoryTest/session1.json

```
{
  "class" : "mil.sstaf.session.control.SessionConfiguration",
  "async" : false
}
```

A.432 SSTAF/src/framework/mil.sstaf.session/src/test/resources/SoldierConfiguration/TestSoldier1.json

```
{
  "class" : "mil.sstaf.core.entity.Soldier",
  "name": "Test Dude",
  "rank": "E6",
  "features": [
  ],
  "configurations": {}
}
```

A.433 SSTAF/src/framework/mil.sstaf.session/src/test/resources/SoldierConfiguration/TestSoldier2.json

```
{
  "class" : "mil.sstaf.core.entity.Soldier",
  "name": "Test Dude 2",
  "rank": "E6",
  "features": [
  ],
  "configurations": {}
}
```

A.434 SSTAF/src/framework/mil.sstaf.session/src/test/resources/SoldierConfiguration/TestSoldier6.json

```
{
  "class" : "mil.sstaf.core.entity.Soldier",
  "name": "Test Dude",
  "rank": "E4",
  "features": [],
  "configurations": {}
}
```

A.435 SSTAF/src/framework/mil.sstaf.session/src/test/resources/UnitConfiguration/Company.json

```
{
  "class" : "mil.sstaf.core.entity.Unit",
  "name": "Test Platoon",
  "type": "Platoon",
  "features": [],
  "configurations": {},
  "soldiers": [
    {
      "position": "CC",
      "soldier": "../SoldierConfiguration/TestSoldier1.json"
    },
    {
      "position": "DCC",
      "soldier": "../SoldierConfiguration/TestSoldier1.json"
    },
    {
      "position": "G2",
      "soldier": "../SoldierConfiguration/TestSoldier1.json"
    }
  ],
  "subUnits": [
    {
      "label": "1st Platoon",
      "unit": "Platoon.json"
    },
    {
      "label": "2nd Platoon",
      "unit": "Platoon.json"
    },
    {
      "label": "3rd Platoon",
      "unit": "Platoon.json"
    }
  ]
}
```

A.436 SSTAF/src/framework/mil.sstaf.session/src/test/resources/UnitConfiguration/FireTeam.json

```
{
  "class" : "mil.sstaf.core.entity.Unit",
  "name": "Test Unit",
  "type": "FireTeam",
  "features": [
  ],
  "configurations": {},
  "soldiers": [
    {
      "position": "TL",
      "soldier": "../SoldierConfiguration/TestSoldier1.json"
    },
    {
      "position": "R",
      "soldier": "../SoldierConfiguration/TestSoldier2.json"
    },
    {
      "position": "G",
      "soldier": "../SoldierConfiguration/TestSoldier1.json"
    },
    {
      "position": "AR",
      "soldier": "../SoldierConfiguration/TestSoldier2.json"
    }
  ]
}
```

A.437 SSTAF/src/framework/mil.sstaf.session/src/test/resources/UnitConfiguration/Platoon.json

```
{
  "class" : "mil.sstaf.core.entity.Unit",
  "name": "Test Platoon",
  "type": "Platoon",
  "features": [],
  "configurations": {},
  "soldiers": [
    {
      "position": "PL",
      "soldier": "../SoldierConfiguration/TestSoldier1.json"
    },
    {
      "position": "DPL",
      "soldier": "../SoldierConfiguration/TestSoldier1.json"
    }
  ],
  "subUnits": [
    {
      "label": "Squad A",
      "unit": "Squad.json"
    },
    {
      "label": "Squad B",
      "unit": "Squad.json"
    },
    {
      "label": "Squad C",
      "unit": "Squad.json"
    }
  ]
}
```

A.438 SSTAF/src/framework/mil.sstaf.session/src/test/resources/UnitConfiguration/Squad.json

```
{
  "class" : "mil.sstaf.core.entity.Unit",
  "name": "Test Squad",
  "type": "Squad",
  "features": [
  ],
  "configurations": {},
  "soldiers": [
    {
      "position": "SL",
      "soldier": "../SoldierConfiguration/TestSoldier1.json"
    }
  ],
  "subUnits": [
    {
      "label": "Fire Team Alpha",
      "unit": "FireTeam.json"
    },
    {
      "label": "Fire Team Bravo",
      "unit": "FireTeam.json"
    }
  ]
}
```

A.439 SSTAF/src/framework/mil.sstaf.session/src/test/resources/log4j2-test.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="INFO">
    <Appenders>
        <Console name="Console" target="SYSTEM_OUT">
            <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"/>
        </Console>
    </Appenders>
    <Loggers>
        <Root level="debug">
            <AppenderRef ref="Console"/>
        </Root>
    </Loggers>
</Configuration>
```

A.440 SSTAF/src/settings.gradle

```
/*
 * Specifies the content to build and other root-level settings.
 */
//
// Specifies from where to retrieve the plugins.
//
pluginManagement {
    repositories {
        mavenLocal()
        mavenCentral()
        gradlePluginPortal()
        maven {
            name = 'localPluginRepository'
            url = System.getProperty('user.home')+"/.m2/local-plugin-repository"
        }
    }
}
rootProject.name = 'Unified SSTAF'
/*
 * Traverse the project and include all of the sub-projects
 */
fileTree('.') {
   include '**/build.gradle'
   exclude 'build.gradle' // Exclude the root build file.
   exclude 'buildSrc/build.gradle' // and the buildSrc dir
}.collect {
    relativePath(it.parent).replace(File.separator, ':')
}.each {
  include(it)
  println("Including ${it}")
}
```

A.441 SSTAF/src/testFeatures/human/mil.sstaftest.anthropometry.api/build.gradle

```
plugins {
    id 'java-library'
}
ext.moduleName = 'mil.sstaftest.anthropometry.api'
dependencies {
    implementation project(':framework:mil.sstaf.core')
    implementation group: 'org.slf4j', name: 'slf4j-api', version: '1.7.32'
    implementation group: 'org.apache.commons', name: 'commons-math3', version: '3.6.1'
}
```

A.442 SSTAF/src/testFeatures/human/mil.sstaftest.anthropometry.api/src/main/java/mil/sstaftest/anthropometry/api/AnthroConfiguration.java

```
package mil.sstaftest.anthropometry.api;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import mil.sstaf.core.features.FeatureConfiguration;
@SuperBuilder
@Jacksonized
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
@EqualsAndHashCode(callSuper = true)
public class AnthroConfiguration  extends FeatureConfiguration {
    @Getter
    private final Sex sex;
    @Getter private final Handedness handedness;
    @Getter private final double height;
    @Getter private final double weight;
    @Getter private final double span;
    @Getter private final int age;
}
```

A.443 SSTAF/src/testFeatures/human/mil.sstaftest.anthropometry.api/src/main/java/mil/sstaftest/anthropometry/api/Anthropometry.java

```
package mil.sstaftest.anthropometry.api;
import mil.sstaf.core.features.Feature;
public interface Anthropometry extends Feature {
    String SEX_KEY = "sex";
    String AGE_KEY = "age";
    String HEIGHT_KEY = "height_cm";
    String SPAN_KEY = "span_cm";
    String WEIGHT_KEY = "weight_kg";
    String HANDEDNESS_KEY = "handedness";
    Sex getSex();
    Handedness getHandedness();
    int getAge();
    double getHeight_cm();
    double getSpan_cm();
    double getWeight_kg();

}
```

A.444 SSTAF/src/testFeatures/human/mil.sstaftest.anthropometry.api/src/main/java/mil/sstaftest/anthropometry/api/Handedness.java

```
package mil.sstaftest.anthropometry.api;
public enum Handedness {
    LEFT,
    RIGHT
}
```

A.445 SSTAF/src/testFeatures/human/mil.sstaftest.anthropometry.api/src/main/java/mil/sstaftest/anthropometry/api/Sex.java

```
package mil.sstaftest.anthropometry.api;
public enum Sex {
    FEMALE(1),
    MALE(2);
    int aVal;
    Sex(int a) {
        aVal = a;
    }
}
```

A.446 SSTAF/src/testFeatures/human/mil.sstaftest.anthropometry.api/src/main/java/module-info.java

```
module mil.sstaftest.anthropometry.api {
    requires mil.sstaf.core;
    exports mil.sstaftest.anthropometry.api;
}
```

A.447 SSTAF/src/testFeatures/human/mil.sstaftest.anthropometry.simple/build.gradle

```
plugins {
    id 'java-library'
}
ext.moduleName = 'mil.sstaftest.anthropometry.simple'
dependencies {
    implementation project(':framework:mil.sstaf.core')
    implementation group: 'org.slf4j', name: 'slf4j-api', version: '1.7.32'
    implementation project(':testFeatures:human:mil.sstaftest.anthropometry.api')
    implementation group: 'org.apache.commons', name: 'commons-math3', version: '3.6.1'
    testImplementation project(':verification:mil.sstaftest.util')
}
```

A.448 SSTAF/src/testFeatures/human/mil.sstaftest.anthropometry.simple/src/main/java/mil/sstaftest/anthropometry/simple/AnthroConfig.java

```
package mil.sstaftest.anthropometry.simple;
import mil.sstaf.core.features.BaseFeature;
import mil.sstaf.core.features.FeatureConfiguration;
import mil.sstaf.core.util.SSTAFException;
import mil.sstaftest.anthropometry.api.AnthroConfiguration;
import mil.sstaftest.anthropometry.api.Anthropometry;
import mil.sstaftest.anthropometry.api.Handedness;
import mil.sstaftest.anthropometry.api.Sex;
/**
 * Provides anthropometric configuration values
 */
public class AnthroConfig extends BaseFeature implements Anthropometry {
    public static final String FEATURE_NAME = "Simple Anthropometry";
    public static final int MAJOR_VERSION = 0;
    public static final int MINOR_VERSION = 1;
    public static final int PATCH_VERSION = 0;
    static final Sex DEFAULT_SEX = Sex.MALE;
    static final int DEFAULT_AGE = 20;
    static final double DEFAULT_HEIGHT_CM = 175;
    static final double DEFAULT_SPAN_CM = 175;
    static final double DEFAULT_WEIGHT_KG = 90;
    private Sex sex;
    private Handedness handedness;
    private int age;
    private double height_cm;
    private double span_cm;
    private double weight_kg;
    public AnthroConfig() {
        super(FEATURE_NAME, MAJOR_VERSION, MINOR_VERSION, PATCH_VERSION,
                false, "A simple representation of anthropometry");
        sex = DEFAULT_SEX;
        age = DEFAULT_AGE;
        height_cm = DEFAULT_HEIGHT_CM;
        span_cm = DEFAULT_SPAN_CM;
        weight_kg = DEFAULT_WEIGHT_KG;
        handedness = Handedness.RIGHT;
    }
    @Override
    public Sex getSex() {
        return sex;
    }
    void setSex(Sex sex) {
        this.sex = sex;
    }
    public Handedness getHandedness() {
        return handedness;
    }
    void setHandedness(Handedness handedness) {
        this.handedness = handedness;
    }
    @Override
    public int getAge() {
        return age;
    }
    void setAge(int age) {
        this.age = age;
    }
    @Override
    public double getHeight_cm() {
        return height_cm;
```

```
    }
    void setHeight_cm(double height_cm) {
        this.height_cm = height_cm;
    }
    @Override
    public double getSpan_cm() {
        return span_cm;
    }
    void setSpan_cm(double span_cm) {
        this.span_cm = span_cm;
    }
    @Override
    public double getWeight_kg() {
        return weight_kg;
    }
    void setWeight_kg(double weight_kg) {
        this.weight_kg = weight_kg;
    }
    @Override
    public String toString() {
        return "AnthroConfig{" +
                "sex=" + sex +
                ", age=" + age +
                ", height_cm=" + height_cm +
                ", span_cm=" + span_cm +
                ", weight_kg=" + weight_kg +
                '}';
    }
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        AnthroConfig that = (AnthroConfig) o;
        return age == that.age &&
                Double.compare(that.height_cm, height_cm) == 0 &&
                Double.compare(that.span_cm, span_cm) == 0 &&
                Double.compare(that.weight_kg, weight_kg) == 0 &&
                sex == that.sex;
    }
    @Override
    public void init() throws SSTAFException {
        super.init();
    }
    @Override
    public void configure(FeatureConfiguration configuration) {
        super.configure(configuration);
        if (configuration instanceof AnthroConfiguration) {
            AnthroConfiguration ac = (AnthroConfiguration) configuration;
            setHandedness(ac.getHandedness());
            setSex(ac.getSex());
            setSpan_cm(ac.getSpan());
            setHeight_cm(ac.getHeight());
            setWeight_kg(ac.getWeight());
            setAge(ac.getAge());
        }
    }
}
```

A.449 SSTAF/src/testFeatures/human/mil.sstaftest.anthropometry.simple/src/main/java/module-info.java

```
import mil.sstaf.core.features.Feature;
import mil.sstaftest.anthropometry.api.Anthropometry;
import mil.sstaftest.anthropometry.simple.AnthroConfig;
module mil.sstaftest.anthropometry.simple {
    exports mil.sstaftest.anthropometry.simple;
    requires mil.sstaf.core;
    requires mil.sstaftest.anthropometry.api;

    provides Feature with AnthroConfig;
    provides Anthropometry with AnthroConfig;
    opens mil.sstaftest.anthropometry.simple to mil.sstaf.core;
}
```

A.450 SSTAF/src/testFeatures/human/mil.sstaftest.anthropometry.simple/src/test/java/mil/sstaftest/anthropometry/simple/AnthroConfigTest.java

```
package mil.sstaftest.anthropometry.simple;
import mil.sstaf.core.features.Feature;
import mil.sstaftest.anthropometry.api.AnthroConfiguration;
import mil.sstaftest.anthropometry.api.Anthropometry;
import mil.sstaftest.anthropometry.api.Sex;
import org.junit.jupiter.api.Test;
import static mil.sstaftest.anthropometry.simple.AnthroConfig.*;
import static org.junit.jupiter.api.Assertions.*;
class AnthroConfigTest {
    @Test
    void emptyBuildYieldsDefaults() {
        var b = new AnthroConfig();
        assertEquals(DEFAULT_SEX, b.getSex());
        assertEquals(DEFAULT_AGE, b.getAge());
        assertEquals(DEFAULT_HEIGHT_CM, b.getHeight_cm(), 0.00001);
        assertEquals(DEFAULT_SPAN_CM, b.getSpan_cm(), 0.00001);
        assertEquals(DEFAULT_WEIGHT_KG, b.getWeight_kg(), 0.00001);
    }
    @Test
    void settingSexWorks() {
        AnthroConfig b = new AnthroConfig();
        b.setSex(Sex.FEMALE);
        assertEquals(Sex.FEMALE, b.getSex());
        assertEquals(DEFAULT_AGE, b.getAge());
        assertEquals(DEFAULT_HEIGHT_CM, b.getHeight_cm(), 0.00001);
        assertEquals(DEFAULT_SPAN_CM, b.getSpan_cm(), 0.00001);
        assertEquals(DEFAULT_WEIGHT_KG, b.getWeight_kg(), 0.00001);
    }
    @Test
    void settingAgeWorks() {
        AnthroConfig b = new AnthroConfig();
        b.setAge(33);
        assertEquals(DEFAULT_SEX, b.getSex());
        assertEquals(33, b.getAge());
        assertEquals(DEFAULT_HEIGHT_CM, b.getHeight_cm(), 0.00001);
        assertEquals(DEFAULT_SPAN_CM, b.getSpan_cm(), 0.00001);
        assertEquals(DEFAULT_WEIGHT_KG, b.getWeight_kg(), 0.00001);
    }
    @Test
    void settingHeightWorks() {
        AnthroConfig b = new AnthroConfig();
        b.setHeight_cm(180);
        assertEquals(DEFAULT_SEX, b.getSex());
        assertEquals(DEFAULT_AGE, b.getAge());
        assertEquals(180, b.getHeight_cm(), 0.00001);
        assertEquals(DEFAULT_SPAN_CM, b.getSpan_cm(), 0.00001);
        assertEquals(DEFAULT_WEIGHT_KG, b.getWeight_kg(), 0.00001);
    }
    @Test
    void settingSpanWorks() {
        AnthroConfig b = new AnthroConfig();
        b.setSpan_cm(170);
        assertEquals(DEFAULT_SEX, b.getSex());
        assertEquals(DEFAULT_AGE, b.getAge());
        assertEquals(DEFAULT_HEIGHT_CM, b.getHeight_cm(), 0.00001);
        assertEquals(170, b.getSpan_cm(), 0.00001);
        assertEquals(DEFAULT_WEIGHT_KG, b.getWeight_kg(), 0.00001);
    }
    @Test
    void settingWeightWorks() {
```

```
        AnthroConfig b = new AnthroConfig();
        b.setWeight_kg(100);
        assertEquals(DEFAULT_SEX, b.getSex());
        assertEquals(DEFAULT_AGE, b.getAge());
        assertEquals(DEFAULT_HEIGHT_CM, b.getHeight_cm(), 0.00001);
        assertEquals(DEFAULT_SPAN_CM, b.getSpan_cm(), 0.00001);
        assertEquals(100, b.getWeight_kg(), 0.00001);
    }
    @Test
    void settingEverythingWorks() {
        AnthroConfig b = new AnthroConfig();
        b.setSex(Sex.FEMALE);
        b.setWeight_kg(45);
        b.setHeight_cm(160);
        b.setSpan_cm(155);
        b.setAge(18);
        assertEquals(Sex.FEMALE, b.getSex());
        assertEquals(18, b.getAge());
        assertEquals(160, b.getHeight_cm(), 0.00001);
        assertEquals(155, b.getSpan_cm(), 0.00001);
        assertEquals(45, b.getWeight_kg(), 0.00001);
        String ts = b.toString();
        assertTrue(ts.contains("age"));
        assertTrue(ts.contains("sex"));
        assertTrue(ts.contains("weight_kg"));
    }
    @Test
    void equalsWorksAsExpected() {
        var a = new AnthroConfig();
        var b = new AnthroConfig();
        assertEquals(a, b);
        b.setAge(31);
        assertNotEquals(a, b);
        a.setAge(31);
        assertEquals(a, b);
        b.setSex(Sex.FEMALE);
        assertNotEquals(a, b);
        a.setSex(Sex.FEMALE);
        assertEquals(a, b);
        b.setHeight_cm(111);
        assertNotEquals(a, b);
        a.setHeight_cm(111);
        assertEquals(a, b);
        b.setSpan_cm(111);
        assertNotEquals(a, b);
        a.setSpan_cm(111);
        assertEquals(a, b);
        b.setWeight_kg(111);
        assertNotEquals(a, b);
        a.setWeight_kg(111);
        assertEquals(a, b);

        Anthropometry defaultAnthro = new AnthroConfig();
        assertEquals(defaultAnthro, defaultAnthro);
        // a little helper class for our next equality test
        final class AlmostAnAnthro extends AnthroConfig {
            AlmostAnAnthro() {
            }
        }
        AlmostAnAnthro wrongClass = new AlmostAnAnthro();
        assertNotEquals(defaultAnthro, wrongClass);
        Anthropometry anotherDefaultAnthro = new AnthroConfig();
        assertEquals(defaultAnthro, anotherDefaultAnthro);
    }
    @Test
    void configureWorks() {
        AnthroConfiguration ac = AnthroConfiguration.builder()
                .sex(Sex.FEMALE)
```

```
                .span(123)
                .weight(450)
                .age(45)
                .height(111).build();
        Anthropometry ap = new AnthroConfig();
        ap.configure(ac);
        assertEquals(Sex.FEMALE, ap.getSex());
        assertEquals(45, ap.getAge());
        assertEquals(111, ap.getHeight_cm());
        assertEquals(123, ap.getSpan_cm());
        assertEquals(450, ap.getWeight_kg());
    }

    @Test
    void featureBehaviorWorksAsExpected() {
        Feature feature = new AnthroConfig();
        assertEquals("Simple Anthropometry", feature.getName());
        assertEquals(0, feature.getMajorVersion());
        assertEquals(1, feature.getMinorVersion());
        assertEquals(0, feature.getPatchVersion());
        assertNotNull(feature.getDescription());
    }
}
```

A.451 SSTAF/src/testFeatures/human/mil.sstaftest.anthropometry.simple/src/test/resources/log4j2-test.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="INFO">
    <Appenders>
        <Console name="Console" target="SYSTEM_OUT">
            <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"/>
        </Console>
    </Appenders>
    <Loggers>
        <Root level="debug">
            <AppenderRef ref="Console"/>
        </Root>
    </Loggers>
</Configuration>
```

A.452 SSTAF/src/testFeatures/integration/mil.sstaftest.alpha/build.gradle

```
dependencies {
    implementation project(':framework:mil.sstaf.core')
    implementation group: 'org.apache.commons', name: 'commons-math3', version: '3.6.1'
    implementation group: 'org.slf4j', name: 'slf4j-api', version: '1.7.32'
}
task copyJar(type: Copy) {
    from jar
    into "../../../../src/framework/mil.sstaf.core/src/integrationTest/resources/modules/jamesbon
d"
}
build.dependsOn copyJar
```

A.453 SSTAF/src/testFeatures/integration/mil.sstaftest.alpha/src/main/java/mil/sstaftest/alpha/AlphaProvider.java

```
package mil.sstaftest.alpha;
import mil.sstaf.core.features.BaseFeature;
import mil.sstaf.core.features.FeatureConfiguration;
import mil.sstaf.core.util.SSTAFException;
public class AlphaProvider extends BaseFeature {
    @SuppressWarnings("unused")
    private FeatureConfiguration configuration;
    public AlphaProvider() {
        super("Alpha", 5, 3, 0, true, "");
    }
    @Override
    public void configure(FeatureConfiguration configuration) {
        super.configure(configuration);
        this.configuration = configuration;
    }
    @Override
    public void init() throws SSTAFException {
        super.init();
        if (configuration == null) {
            throw new SSTAFException("Null configuration");
        }
    }
}
```

A.454 SSTAF/src/testFeatures/integration/mil.sstaftest.alpha/src/main/java/module-info.java

```
module mil.sstaftest.alpha {
    exports mil.sstaftest.alpha;
    requires mil.sstaf.core;

    provides mil.sstaf.core.features.Feature with mil.sstaftest.alpha.AlphaProvider;
    opens mil.sstaftest.alpha to mil.sstaf.core;
}
```

A.455 SSTAF/src/testFeatures/integration/mil.sstaftest.bravo/build.gradle

```
dependencies {
    implementation project(':framework:mil.sstaf.core')
    implementation group: 'org.apache.commons', name: 'commons-math3', version: '3.6.1'
}
task copyJar(type: Copy) {
    from jar
    into "../../../../src/framework/mil.sstaf.core/src/integrationTest/resources/modules/jamesbon
d"
}
build.dependsOn copyJar
```

A.456 SSTAF/src/testFeatures/integration/mil.sstaftest.bravo/src/main/java/mil/sstaftest/bravo/BravoProvider.java

```
package mil.sstaftest.bravo;
import mil.sstaf.core.features.BaseFeature;
import mil.sstaf.core.features.Feature;
import mil.sstaf.core.features.FeatureConfiguration;
import mil.sstaf.core.features.Requires;
import mil.sstaf.core.util.SSTAFException;
import java.util.Objects;
public class BravoProvider extends BaseFeature {
    @Requires(name = "Charlie", majorVersion = 1, minorVersion = 2, requireExact = true)
    private Feature charlieProvider;
    @Requires(name = "Delta", majorVersion = 8)
    private Feature deltaProvider;
    private FeatureConfiguration configuration;
    public BravoProvider() {
        super("Bravo", 3, 4, 0, true, "");
    }
    @Override
    public void init() throws SSTAFException {
        super.init();
        Objects.requireNonNull(charlieProvider, "No Charlie");
        Objects.requireNonNull(deltaProvider, "No Delta");
        Objects.requireNonNull(configuration, "No MapConfiguration");
        charlieProvider.init();
        deltaProvider.init();
    }
    @Override
    public void configure(FeatureConfiguration configuration) {
        super.configure(configuration);
        this.configuration = configuration;
    }
}
```

A.457 SSTAF/src/testFeatures/integration/mil.sstaftest.bravo/src/main/java/module-info.java

```
module mil.sstaftest.bravo {
    exports mil.sstaftest.bravo;
    requires mil.sstaf.core;

    provides mil.sstaf.core.features.Feature with mil.sstaftest.bravo.BravoProvider;
    opens mil.sstaftest.bravo to mil.sstaf.core;
}
```

A.458 SSTAF/src/testFeatures/integration/mil.sstaftest.charlie/build.gradle

```
dependencies {
    implementation project(':framework:mil.sstaf.core')
    implementation group: 'org.apache.commons', name: 'commons-math3', version: '3.6.1'
}
task copyJar(type: Copy) {
    from jar
    into "../../../../src/framework/mil.sstaf.core/src/integrationTest/resources/modules/jamesbon
d"
}
build.dependsOn copyJar
```

A.459 SSTAF/src/testFeatures/integration/mil.sstaftest.charlie/src/main/java/mil/sstaftest/charlie/CharlieProvider.java

```
package mil.sstaftest.charlie;
import mil.sstaf.core.features.BaseFeature;
import mil.sstaf.core.features.FeatureConfiguration;
import mil.sstaf.core.features.Requires;
import mil.sstaf.core.util.SSTAFException;
public class CharlieProvider extends BaseFeature {
    private FeatureConfiguration configuration;
    @Requires
    private Runnable echo;
    public CharlieProvider() {
        super("Charlie", 1, 2, 0, true, "");
    }
    @Override
    public void init() throws SSTAFException {
        super.init();
        if (configuration == null) {
            throw new SSTAFException("Null configuration");
        }
    }
    @Override
    public void configure(FeatureConfiguration configuration) {
        super.configure(configuration);
        this.configuration = configuration;
    }
}
```

A.460 SSTAF/src/testFeatures/integration/mil.sstaftest.charlie/src/main/java/module-info.java

```
module mil.sstaftest.charlie {
    exports mil.sstaftest.charlie;
    requires mil.sstaf.core;

    provides mil.sstaf.core.features.Feature with mil.sstaftest.charlie.CharlieProvider;
}
```

A.461 SSTAF/src/testFeatures/integration/mil.sstaftest.delta/build.gradle

```
dependencies {
    implementation project(':framework:mil.sstaf.core')
    implementation group: 'org.apache.commons', name: 'commons-math3', version: '3.6.1'
}
task copyJar(type: Copy) {
    from jar
    into "../../../../src/framework/mil.sstaf.core/src/integrationTest/resources/modules/jamesbon
d"
}
build.dependsOn copyJar
```

A.462 SSTAF/src/testFeatures/integration/mil.sstaftest.delta/src/main/java/mil/sstaftest/delta/DeltaProvider.java

```
package mil.sstaftest.delta;
import mil.sstaf.core.features.BaseFeature;
import mil.sstaf.core.features.FeatureConfiguration;
import mil.sstaf.core.util.SSTAFException;
public class DeltaProvider extends BaseFeature {
    FeatureConfiguration configuration;
    public DeltaProvider() {
        super("Delta", 8, 9, 0, true, "");
    }
    @Override
    public void init() throws SSTAFException {
        super.init();
        if (configuration == null) {
            throw new SSTAFException("Null configuration");
        }
    }
    @Override
    public void configure(FeatureConfiguration configuration) {
        super.configure(configuration);
        this.configuration = configuration;
    }
}
```

A.463 SSTAF/src/testFeatures/integration/mil.sstaftest.delta/src/main/java/module-info.java

```
module mil.sstaftest.delta {
    exports mil.sstaftest.delta;
    requires mil.sstaf.core;

    provides mil.sstaf.core.features.Feature with mil.sstaftest.delta.DeltaProvider;
    opens mil.sstaftest.delta to mil.sstaf.core;
}
```

A.464 SSTAF/src/testFeatures/integration/mil.sstaftest.echo/build.gradle

```
dependencies {
    implementation project(':framework:mil.sstaf.core')
    implementation group: 'org.apache.commons', name: 'commons-math3', version: '3.6.1'
}
task copyJar(type: Copy) {
    from jar
    into "../../../../src/framework/mil.sstaf.core/src/integrationTest/resources/modules/jamesbon
d"
}
build.dependsOn copyJar
```

A.465 SSTAF/src/testFeatures/integration/mil.sstaftest.echo/src/main/java/mil/sstaftest/echo/EchoProvider.java

```
package mil.sstaftest.echo;
import lombok.Getter;
import lombok.experimental.SuperBuilder;
import mil.sstaf.core.features.*;
import mil.sstaf.core.util.SSTAFException;
public class EchoProvider extends BaseFeature implements Runnable {
    @Requires(name = "Bravo", majorVersion = 3, minorVersion = 4, requireExact = true)
    private Feature bravo;
    public EchoProvider() {
        super("Echo", 1, 0, 0, true, "");
    }
    @Override
    public void init() throws SSTAFException {
        super.init();
    }
    @Override
    public void configure(FeatureConfiguration configuration) {
        super.configure(configuration);
    }
    @Override
    public void run() {
    }
}
```

A.466 SSTAF/src/testFeatures/integration/mil.sstaftest.echo/src/main/java/module-info.java

```
module mil.sstaftest.echo {
    exports mil.sstaftest.echo;
    requires mil.sstaf.core;

    provides Runnable with mil.sstaftest.echo.EchoProvider;
    opens mil.sstaftest.echo to mil.sstaf.core;
}
```

A.467 SSTAF/src/testFeatures/integration/mil.sstaftest.jamesbond/build.gradle

```
dependencies {
    implementation project(':framework:mil.sstaf.core')
    implementation group: 'org.apache.commons', name: 'commons-math3', version: '3.6.1'
}
task copyJar(type: Copy) {
    from jar
    into "../../../../src/framework/mil.sstaf.core/src/integrationTest/resources/modules/jamesbon
d"
}
build.dependsOn copyJar
```

A.468 SSTAF/src/testFeatures/integration/mil.sstaftest.jamesbond/src/main/java/mil/sstaftest/jamesbond/JamesBond.java

```
package mil.sstaftest.jamesbond;
import mil.sstaf.core.entity.Address;
import mil.sstaf.core.entity.EntityHandle;
import mil.sstaf.core.features.*;
import mil.sstaf.core.util.SSTAFException;
import java.util.List;
import java.util.Objects;
public class JamesBond extends BaseAgent {
    private FeatureConfiguration configuration;
    @Requires(name = "Alpha", majorVersion = 5, minorVersion = 3, requireExact = true)
    private Feature alphaProvider;
    @Requires(name = "Bravo", majorVersion = 2, minorVersion = 1)
    private Feature bravoProvider;
    public JamesBond() {
        super("James Bond", 7, 0, 0, true,
                "Licensed to throw exceptions");
        ownerHandle = EntityHandle.makeDummyHandle();
    }
    @Override
    public ProcessingResult tick(long currentTime_ms) {
        return null;
    }
    @Override
    public List<Class<? extends HandlerContent>> contentHandled() {
        return List.of();
    }
    @Override
    public void init() {
        super.init();
        if (configuration == null) {
            throw new SSTAFException("Null configuration");
        }
        Objects.requireNonNull(alphaProvider, "No Alpha");
        Objects.requireNonNull(bravoProvider, "No Bravo");
        alphaProvider.init();
        bravoProvider.init();
    }
    @Override
    public ProcessingResult process(HandlerContent arg, long scheduledTime_ms,
        long currentTime_ms, Address from, long id, Address respondTo) {
        return buildUnsupportedMessageResponse(arg, id, getAddress(), new Throwable());
    }
    @Override
    public void configure(FeatureConfiguration configuration) {
        super.configure(configuration);
        this.configuration =  configuration;
    }
}
```

A.469 SSTAF/src/testFeatures/integration/mil.sstaftest.jamesbond/src/main/java/module-info.java

```
module mil.sstaftest.jamesbond {
    exports mil.sstaftest.jamesbond;
    requires mil.sstaf.core;

    provides mil.sstaf.core.features.Feature with mil.sstaftest.jamesbond.JamesBond;
    opens mil.sstaftest.jamesbond to mil.sstaf.core;
}
```

A.470 SSTAF/src/testFeatures/integration/mil.sstaftest.simplemock/build.gradle

```
dependencies {
    implementation project(':framework:mil.sstaf.core')
    implementation group: 'org.apache.commons', name: 'commons-math3', version: '3.6.1'
}
sourceSets {
    main {
        resources {
            srcDirs "src/main/resources", "src/main/python"
        }
    }
}
jar {
    manifest {
        def fileList = []
        ['src/main/python', 'src/main/resource'].each { String dir ->
            fileTree(dir) {
                include '**/*.py'
                include '**/*.json'
            }.collect {
                // println(it.getPath())
                relativePath(it)
            }.each {
                // println(it)
                def trimmed = it.replace("$dir/",'')
                fileList << trimmed
                // println(fileList)
            }
        }
        attributes('SSTAF-Module' : 'mil.sstaftest.simplemock')
        attributes('SSTAF-Resources-To-Extract' : fileList.join(' '))
    }
}
```

A.471 SSTAF/src/testFeatures/integration/mil.sstaftest.simplemock/src/main/java/mil/sstaftest/simplemock/SimpleMockFeature.java

```
package mil.sstaftest.simplemock;
import mil.sstaf.core.features.BaseFeature;
public class SimpleMockFeature extends BaseFeature {
    /**
     * Constructor for subclasses.
     * <p>
     * Note that concrete implementations must have a no-args constructor to be loadable as servi
ces
     */
    protected SimpleMockFeature() {
        super("Simple Mock Feature",
                1, 0, 0,
                false, "This is nothing");
    }
}
```

A.472 SSTAF/src/testFeatures/integration/mil.sstaftest.simplemock/src/main/java/module-info.java

```
module mil.sstaftest.simplemock {
    exports mil.sstaftest.simplemock;
    requires mil.sstaf.core;
    //
    // Required to make the scripts accessible
    //
    opens mil.sstaftest.simplemock to mil.sstaf.core;
}
```

A.473 SSTAF/src/testFeatures/integration/mil.sstaftest.simplemock/src/main/python/mil/sstaftest/simplemock/args.py

```
import sys
s = sys.stdin.readline().strip()
while s not in ['break', 'quit']:
    index = int(s)
    sys.stdout.write(sys.argv[index] + '\n')
    sys.stdout.flush()
    s = sys.stdin.readline().strip()
```

A.474 SSTAF/src/testFeatures/integration/mil.sstaftest.simplemock/src/main/python/mil/sstaftest/simplemock/upper.py

```
import sys
s = sys.stdin.readline().strip()
while s not in ['break', 'quit']:
    sys.stdout.write(s.upper() + '\n')
    sys.stdout.flush()
    s = sys.stdin.readline().strip()
```

A.475 SSTAF/src/testFeatures/military/mil.sstaftest.maneuver.api/build.gradle

```
plugins {
    id 'java-library'
}
ext.moduleName = 'mil.sstaftest.maneuver.api'
dependencies {
    implementation project(':framework:mil.sstaf.core')
}
```

A.476 SSTAF/src/testFeatures/military/mil.sstaftest.maneuver.api/src/main/java/mil/sstaftest/maneuver/api/Heading.java

```
package mil.sstaftest.maneuver.api;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.EqualsAndHashCode;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import mil.sstaf.core.features.HandlerContent;
@SuperBuilder
@Jacksonized
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
@EqualsAndHashCode(callSuper = true)
public class Heading extends HandlerContent {
    public final double heading_degs;
    public final double heading_rads;
    private Heading(final double headingDegrees) {
        this.heading_degs = headingDegrees;
        double constrained = (heading_degs <= 180.0) ? heading_degs : heading_degs - 360.0;
        double flipped = 90.0 - constrained;
        heading_rads = flipped * Math.PI / 180.0;
    }
    public static Heading of(final double headingDeg) {
        if (headingDeg < 0.0 || headingDeg > 360.0) {
            throw new IllegalArgumentException("Heading must be in the range [0.0,360.0]");
        }
        return new Heading(headingDeg);
    }
    @Override
    public String toString() {
        return "Heading{" +
                "heading_degs=" + heading_degs +
                ", heading_rads=" + heading_rads +
                '}';
    }
}
```

A.477 SSTAF/src/testFeatures/military/mil.sstaftest.maneuver.api/src/main/java/mil/sstaftest/maneuver/api/ManeuverState.java

```
package mil.sstaftest.maneuver.api;
import lombok.Data;
import lombok.EqualsAndHashCode;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import mil.sstaf.core.entity.EntityHandle;
import mil.sstaf.core.features.HandlerContent;
import mil.sstaf.core.state.State;
import mil.sstaf.core.state.StateProperty;
import java.util.Objects;
/**
 * Immutable description of the current position, heading and velocity of the Entity.
 */
@Data
@EqualsAndHashCode(callSuper = true)
@SuperBuilder
@Jacksonized
public class ManeuverState extends HandlerContent implements State {
    public final String path;
    public final long timestamp_ms;
    public final Position position;
    public final Heading heading;
    public final Speed speed;
    @StateProperty(headerLabel = "position")
    public Position getPosition() {
        return position;
    }
    @StateProperty(headerLabel = "heading")
    public Heading getHeading() {
        return heading;
    }
    @StateProperty(headerLabel =  "speed")
    public Speed getSpeed() {
        return speed;
    }
    /**
     * Creates a ManeuverState
     *
     * @param owner        the EntityHandle for the Entity that produced this record.
     * @param timestamp_ms the simulation time at which the record was created
     * @param position     the Position of the Entity at the time specified by the timestamp
     * @param heading      the Heading of the Entity
     * @param speed        the Speed of the Entity
     * @return a new ManeuverState
     */
    public static ManeuverState of(EntityHandle owner, final long timestamp_ms, final Position po
sition, final Heading heading, final Speed speed) {
        Objects.requireNonNull(owner, "Owner must not be null");
        Objects.requireNonNull(position, "Position must not be null");
        Objects.requireNonNull(heading, "Heading must not be null");
        Objects.requireNonNull(speed, "Speed must not be null");
        ManeuverState.ManeuverStateBuilder<?,?> builder = builder();
        builder.path(owner.getPath());
        builder.timestamp_ms(timestamp_ms);
        builder.position(position);
        builder.heading(heading);
        builder.speed(speed);
        return builder.build();
    }
}
```

A.478 SSTAF/src/testFeatures/military/mil.sstaftest.maneuver.api/src/main/java/mil/sstaftest/maneuver/api/ManeuverStateMap.java

```
package mil.sstaftest.maneuver.api;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.EqualsAndHashCode;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import mil.sstaf.core.features.HandlerContent;
import java.util.Collections;
import java.util.HashMap;
import java.util.Map;
import java.util.Objects;
@SuperBuilder
@Jacksonized
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
@EqualsAndHashCode(callSuper = true)
public class ManeuverStateMap extends HandlerContent {
    private final Map<String, ManeuverState> stateMap = new HashMap<>();
    public void addManeuverState(final ManeuverState maneuverState) {
        Objects.requireNonNull(maneuverState, "ManeuverState was null");
        stateMap.put(maneuverState.path, maneuverState);
    }
    public Map<String, ManeuverState> getStateMap() {
        return Collections.unmodifiableMap(stateMap);
    }
}
```

A.479 SSTAF/src/testFeatures/military/mil.sstaftest.maneuver.api/src/main/java/mil/sstaftest/maneuver/api/ManeuverStateQuery.java

```
package mil.sstaftest.maneuver.api;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.EqualsAndHashCode;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import mil.sstaf.core.features.HandlerContent;
@SuperBuilder
@Jacksonized
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
@EqualsAndHashCode(callSuper = true)
public class ManeuverStateQuery extends HandlerContent {
}
```

A.480 SSTAF/src/testFeatures/military/mil.sstaftest.maneuver.api/src/main/java/mil/sstaftest/maneuver/api/Movable.java

```
package mil.sstaftest.maneuver.api;
public interface Movable {
    Position getPosition();
    void setPosition(Position position);
    Heading getHeading();
    void setHeading(Heading direction);
    double getSpeed();
    void setSpeed(double speed_mps);
    void tick(double deltaT);
    double getDistanceTo(Movable other);
}
```

A.481 SSTAF/src/testFeatures/military/mil.sstaftest.maneuver.api/src/main/java/mil/sstaftest/maneuver/api/Position.java

```
package mil.sstaftest.maneuver.api;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.EqualsAndHashCode;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import mil.sstaf.core.features.HandlerContent;
@SuperBuilder
@Jacksonized
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
@EqualsAndHashCode(callSuper = true)
public class Position extends HandlerContent {
    public final double x;
    public final double y;
    private Position(double xPos, double yPos) {
        this.x = xPos;
        this.y = yPos;
    }
    public static Position of(final double xPos, final double yPos) {
        return new Position(xPos, yPos);
    }
    public static Position copy(final Position orig) {
        return new Position(orig.x, orig.y);
    }
    public double distanceTo(final Position other) {
        double dX = this.x - other.x;
        double dY = this.y - other.y;
        return Math.sqrt(dX * dX + dY * dY);
    }
    @Override
    public String toString() {
        return "Position{" +
                "x=" + x +
                ", y=" + y +
                '}';
    }
}
```

A.482 SSTAF/src/testFeatures/military/mil.sstaftest.maneuver.api/src/main/java/mil/sstaftest/maneuver/api/Speed.java

```
package mil.sstaftest.maneuver.api;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.EqualsAndHashCode;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import mil.sstaf.core.features.HandlerContent;
@SuperBuilder
@Jacksonized
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
@EqualsAndHashCode(callSuper = true)
public class Speed extends HandlerContent {
    public final double speed_ms;
    private Speed(double speed_ms) {
        this.speed_ms = speed_ms;
    }
    public static Speed of(double speed_ms) {
        return new Speed(speed_ms);
    }
    @Override
    public String toString() {
        return "Speed{" +
                "speed_ms=" + speed_ms +
                '}';
    }
}
```

A.483 SSTAF/src/testFeatures/military/mil.sstaftest.maneuver.api/src/main/java/mil/sstaftest/maneuver/api/provider/ManeuverProvider.java

```
package mil.sstaftest.maneuver.api.provider;
import mil.sstaf.core.features.Agent;
import mil.sstaftest.maneuver.api.Heading;
import mil.sstaftest.maneuver.api.Position;
public interface ManeuverProvider extends Agent {
    Position getPosition();
    void setPosition(Position position);
    Heading getHeading();
    void setHeading(Heading direction);
    double getSpeed();
    void setSpeed(double speed_mps);
}
```

A.484 SSTAF/src/testFeatures/military/mil.sstaftest.maneuver.api/src/main/java/module-info.java

```
module mil.sstaftest.maneuver.api {
    requires mil.sstaf.core;

    requires lombok;
    exports mil.sstaftest.maneuver.api;
    exports mil.sstaftest.maneuver.api.provider;
}
```

A.485 SSTAF/src/testFeatures/military/mil.sstaftest.maneuver.api/src/test/java/mil/sstaftest/maneuver/api/HeadingTest.java

```
package mil.sstaftest.maneuver.api;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertThrows;
class HeadingTest {
    @Test
    void constructionWorks() {
        Heading h = Heading.of(0); // North
        assertEquals(0.0, h.heading_degs);
        assertEquals(Math.PI / 2.0, h.heading_rads);
        h = Heading.of(315); // North West;
        assertEquals(315.0, h.heading_degs);
        assertEquals(0.75 * Math.PI, h.heading_rads);
        h = Heading.of(180); // South;
        assertEquals(180.0, h.heading_degs);
        assertEquals(-Math.PI / 2.0, h.heading_rads);
        h = Heading.of(90); // East;
        assertEquals(90.0, h.heading_degs);
        assertEquals(0.0, h.heading_rads);
        assertThrows(IllegalArgumentException.class, () -> Heading.of(-17));
        assertThrows(IllegalArgumentException.class, () -> Heading.of(1701));
    }
}
```

A.486 SSTAF/src/testFeatures/military/mil.sstaftest.maneuver.api/src/test/java/mil/sstaftest/maneuver/api/PositionTest.java

```
package mil.sstaftest.maneuver.api;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;
class PositionTest {
    @Test
    void allPositionBehaviorsWork() {
        Position a = Position.of(1, 0);
        Position b = Position.of(0, 1);
        Position c = Position.of(-1, 0);
        Position d = Position.of(0, -1.0);
        double sqrt2 = Math.sqrt(2.000000);
        Assertions.assertNotNull(a);
        Assertions.assertNotNull(b);
        Assertions.assertNotNull(c);
        Assertions.assertNotNull(d);
        Assertions.assertEquals(1, a.x);
        Assertions.assertEquals(1, b.y);
        Assertions.assertEquals(-1, c.x);
        Assertions.assertEquals(-1, d.y);
        Assertions.assertEquals(0, a.distanceTo(a));
        Assertions.assertEquals(0, b.distanceTo(b));
        Assertions.assertEquals(0, c.distanceTo(c));
        Assertions.assertEquals(0, d.distanceTo(d));

        Assertions.assertEquals(sqrt2, a.distanceTo(b));
        Assertions.assertEquals(sqrt2, b.distanceTo(c));
        Assertions.assertEquals(sqrt2, c.distanceTo(d));
        Assertions.assertEquals(sqrt2, d.distanceTo(a));
        Assertions.assertEquals(2.0, a.distanceTo(c));
        Assertions.assertEquals(2.0, c.distanceTo(a));
    }
}
```

A.487 SSTAF/src/testFeatures/military/mil.sstaftest.maneuver.centralagent/build.gradle

```
plugins {
    id 'java-library'
}
ext.moduleName = 'mil.sstaftest.maneuver.centralagent'
dependencies {
    implementation project(':framework:mil.sstaf.core')
    implementation project(':features:support:mil.sstaf.blackboard.api')
    implementation project(':testFeatures:military:mil.sstaftest.maneuver.api')
    testImplementation project(':verification:mil.sstaftest.util')
}
```

A.488 SSTAF/src/testFeatures/military/mil.sstaftest.maneuver.centralagent/src/main/java/mil/sstaftest/maneuver/centralagent/ManeuverCentralAgent.java

```
package mil.sstaftest.maneuver.centralagent;
import mil.sstaf.blackboard.api.Blackboard;
import mil.sstaf.core.entity.*;
import mil.sstaf.core.features.*;
import mil.sstaf.core.entity.Message;
import mil.sstaf.core.util.Injected;
import mil.sstaf.core.util.SSTAFException;
import mil.sstaftest.maneuver.api.ManeuverState;
import mil.sstaftest.maneuver.api.ManeuverStateMap;
import java.util.ArrayList;
import java.util.List;
import java.util.Optional;
public class ManeuverCentralAgent extends BaseAgent {
    public static final String FEATURE_NAME = "Maneuver Central Agent";
    public static final int MAJOR_VERSION = 0;
    public static final int MINOR_VERSION = 1;
    public static final int PATCH_VERSION = 0;

    @Requires(name = "Blackboard", minorVersion = 1)
    private final Blackboard blackboard = null;
    @Injected
    private EntityRegistry registry;
    private ManeuverStateMap maneuverStateMap = null;
    public ManeuverCentralAgent() {
        super(FEATURE_NAME, MAJOR_VERSION, MINOR_VERSION, PATCH_VERSION,
                false, "Central agent for tracking and communicating positions");
    }
    @Override
    public ProcessingResult tick(long currentTime_ms) {
        List<Message> output = new ArrayList<>();
        if (maneuverStateMap != null) {
            Object EntityHandle;
            for (String path : maneuverStateMap.getStateMap().keySet()) {
                Optional<EntityHandle> oeh = registry.getHandle(path);
                oeh.ifPresent(entityHandle -> output.add(prepareMessage(entityHandle)));
            }
        }
        maneuverStateMap = null;
        return ProcessingResult.of(output);
    }
    private Message prepareMessage(final EntityHandle destination) {
        var builder = EntityAction.builder();
        builder.source(Address.makeAddress(ownerHandle, getName()));
        builder.destination(Address.makeExternalAddress(destination));
        builder.content(maneuverStateMap);
        return builder.build();
    }
    @Override
    public List<Class<? extends HandlerContent>> contentHandled() {
        return List.of(ManeuverState.class);
    }
    @Override
    public void init() {
        super.init();
        if (registry == null) {
            throw new IllegalStateException("EntityRegistry has not been injected");
        }
    }
    @Override
```

```
    public ProcessingResult process(HandlerContent arg, long scheduledTime_ms, long currentTime_m
s,
                                    Address from, long id, Address respondTo) {
        if (arg instanceof ManeuverState) {
            ManeuverState maneuverState = (ManeuverState) arg;
            if (maneuverStateMap == null) {
                maneuverStateMap = ManeuverStateMap.builder().build();
            }
            maneuverStateMap.addManeuverState(maneuverState);
        } else {
            throw new SSTAFException(arg.getClass() + " is not supported by this Handler");
        }
        return ProcessingResult.empty();
    }
}
```

A.489 SSTAF/src/testFeatures/military/mil.sstaftest.maneuver.centralagent/src/main/java/module-info.java

```
import mil.sstaf.core.features.Agent;
import mil.sstaf.core.features.Feature;
import mil.sstaf.core.features.Handler;
import mil.sstaftest.maneuver.centralagent.ManeuverCentralAgent;
module mil.sstaftest.maneuver.centralagent {
    requires mil.sstaf.core;
    requires mil.sstaftest.maneuver.api;

    requires mil.sstaf.blackboard.api;
    exports mil.sstaftest.maneuver.centralagent;
    opens mil.sstaftest.maneuver.centralagent
            to mil.sstaf.core;
    provides Feature with ManeuverCentralAgent;
    provides Handler with ManeuverCentralAgent;
    provides Agent with ManeuverCentralAgent;
}
```

A.490 SSTAF/src/testFeatures/military/mil.sstaftest.maneuver.entityagent/build.gradle

```
dependencies {
    implementation project(':framework:mil.sstaf.core')
    implementation project(':testFeatures:military:mil.sstaftest.maneuver.api')
    implementation project(':features:support:mil.sstaf.blackboard.api')
}
testing {
    suites {
        integrationTest {
            dependencies {
                implementation project(':framework:mil.sstaf.core')
                implementation project(':testFeatures:military:mil.sstaftest.maneuver.api')
                implementation project(':features:support:mil.sstaf.blackboard.api')
                implementation project(':testFeatures:military:mil.sstaftest.maneuver.api')
                implementation project(':testFeatures:military:mil.sstaftest.maneuver.entityagent
')
                implementation project(':features:support:mil.sstaf.blackboard.inmem')
                implementation project(':verification:mil.sstaftest.util')
                runtimeOnly project(':features:support:mil.sstaf.blackboard.inmem')
                runtimeOnly project(':testFeatures:military:mil.sstaftest.maneuver.entityagent')
            }
        }
    }
}
```

A.491 SSTAF/src/testFeatures/military/mil.sstaftest.maneuver.entityagent/src/integrationTest/java/mil/sstaftest/maneuver/entityagent/integration/ManeuverEntityAgentTest.java

```
package mil.sstaftest.maneuver.entityagent.integration;
import lombok.experimental.SuperBuilder;
import mil.sstaf.blackboard.api.Blackboard;
import mil.sstaf.core.entity.*;
import mil.sstaf.core.features.*;
import mil.sstaf.core.entity.Message;
import mil.sstaf.core.util.Injector;
import mil.sstaf.core.configuration.SSTAFConfiguration;
import mil.sstaftest.maneuver.api.*;
import mil.sstaftest.maneuver.entityagent.ManeuverEntityAgent;
import org.slf4j.LoggerFactory;
import org.slf4j.Logger;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import java.io.File;
import java.util.List;
import static org.junit.jupiter.api.Assertions.*;
class ManeuverEntityAgentTest {
    Logger logger = LoggerFactory.getLogger(ManeuverEntityAgentTest.class);
    private Bob bob;
    private ManeuverEntityAgent agent;
    private Blackboard blackboard;
    @BeforeEach
    void setup() {
        logger.info("Beginning setup");
        System.setProperty(SSTAFConfiguration.SSTAF_CONFIGURATION_PROPERTY,
                "src" + File.separator +
                        "integrationTest" + File.separator +
                        "resources" + File.separator +
                        "EmptyConfiguration.json");
        var bobTheBuilder = Bob.builder();
        bobTheBuilder.name("Stand-in for EntityController");
        bob = bobTheBuilder.build();
        bob.setForce(Force.BLUE);
        FeatureSpecification fs = FeatureSpecification.builder().featureClass(Feature.class)
                .featureName(ManeuverEntityAgent.FEATURE_NAME).build();
        Resolver resolver = Resolver.makeTransientResolver();
        agent = (ManeuverEntityAgent) resolver.loadAndResolveDependencies(fs);
        assertNotNull(agent);
        blackboard = resolver.getServicesFromCache(Blackboard.class).get(0);
        assertNotNull(blackboard);
        blackboard.addEntry("SYSTEM:EntityController", bob.getHandle(), Blackboard.BIGBANG);
        logger.info("Done setup");
        Injector.inject(agent, bob.getHandle());
        Injector.inject(blackboard, bob.getHandle());
    }
    @Test
    void tick() {
        Position p0 = Position.of(0.0, 0.0);
        Heading h0 = Heading.of(90.0);
        Speed s0 = Speed.of(2.0);
        ManeuverState ms = ManeuverState.of(bob.getHandle(), 0, p0, h0, s0);
        agent.process(ms, 0, 10000,
                Address.makeExternalAddress(bob.getHandle()), 1,
                Address.makeExternalAddress(bob.getHandle()));
        ProcessingResult output = agent.tick(20000);
        assertEquals(1, output.messages.size());
        Message sm = output.messages.get(0);
        assertTrue(sm instanceof EntityEvent);
        EntityEvent entityEvent = (EntityEvent) sm;
```

```
        Object contents = entityEvent.getContent();
        assertNotNull(contents);
        assertTrue(contents instanceof ManeuverState);
        ManeuverState got = (ManeuverState) contents;
        ManeuverState expected = ManeuverState.of(bob.getHandle(),
                20000, Position.of(40.0, 0.0),
                h0, s0);
        assertEquals(expected, got);
        output = agent.tick(40000);
        assertEquals(1, output.messages.size());
        sm = output.messages.get(0);
        assertTrue(sm instanceof EntityEvent);
        entityEvent = (EntityEvent) sm;
        contents = entityEvent.getContent();
        assertNotNull(contents);
        assertTrue(contents instanceof ManeuverState);
        got = (ManeuverState) contents;
        expected = ManeuverState.of(bob.getHandle(),
                40000, Position.of(80.0, 0.0),
                h0, s0);
        assertEquals(expected, got);
    }

    @Test
    void contentHandled() {
        assertNotNull(agent);
        assertNotNull(blackboard);
        List<Class<? extends HandlerContent>> out = agent.contentHandled();
        assertTrue(out.contains(ManeuverState.class));
        assertTrue(out.contains(ManeuverStateQuery.class));
        assertTrue(out.contains(Position.class));
        assertTrue(out.contains(Heading.class));
        assertTrue(out.contains(Speed.class));
        assertFalse(out.contains(Integer.class));
    }
    @Test
    void process() {
        Position p0 = Position.of(0.0, 0.0);
        Position p1 = Position.of(10.0, 10.0);
        Heading h0 = Heading.of(0.0);
        Heading h1 = Heading.of(90.0);
        Speed s0 = Speed.of(0.0);
        Speed s1 = Speed.of(2.0);
        ManeuverState expected = ManeuverState.of(bob.getHandle(), 0, p0, h0, s0);
        ProcessingResult processingResult = agent.process(ManeuverStateQuery.builder().build(), 0
, 0,
                Address.makeExternalAddress(bob.getHandle()), 1,
                Address.makeExternalAddress(bob.getHandle()));
        assertEquals(1, processingResult.messages.size());
        Message message = processingResult.messages.get(0);
        assertTrue(message.getContent() instanceof ManeuverState);
        ManeuverState ms = (ManeuverState) message.getContent();
        assertEquals(expected, ms);
        expected = ManeuverState.of(bob.getHandle(), 10000, p1, h0, s0);
        processingResult = agent.process(expected, 10000, 10000,
                Address.makeExternalAddress(bob.getHandle()), 1,
                Address.makeExternalAddress(bob.getHandle()));
        assertEquals(1, processingResult.messages.size());
        message = processingResult.messages.get(0);
        assertTrue(message.getContent() instanceof ManeuverState);
        ms = (ManeuverState) message.getContent();
        assertEquals(expected, ms);
        expected = ManeuverState.of(bob.getHandle(), 20000, p1, h1, s0);
        processingResult = agent.process(expected, 20000, 20000,
                Address.makeExternalAddress(bob.getHandle()), 1,
                Address.makeExternalAddress(bob.getHandle()));
        assertEquals(1, processingResult.messages.size());
        message = processingResult.messages.get(0);
```

```
        assertTrue(message.getContent() instanceof ManeuverState);
        ms = (ManeuverState) message.getContent();
        assertEquals(expected, ms);

        expected = ManeuverState.of(bob.getHandle(), 30000, p1, h1, s1);
        processingResult = agent.process(expected, 30000, 30000,
                Address.makeExternalAddress(bob.getHandle()), 1,
                Address.makeExternalAddress(bob.getHandle()));
        assertEquals(1, processingResult.messages.size());
        message = processingResult.messages.get(0);
        assertTrue(message.getContent() instanceof ManeuverState);
        ms = (ManeuverState) message.getContent();
        assertEquals(expected, ms);
        expected = ManeuverState.of(bob.getHandle(), 40000, p0, h0, s0);
        processingResult = agent.process(expected, 40000, 40000,
                Address.makeExternalAddress(bob.getHandle()), 1,
                Address.makeExternalAddress(bob.getHandle()));
        assertEquals(1, processingResult.messages.size());
        message = processingResult.messages.get(0);
        assertTrue(message.getContent() instanceof ManeuverState);
        ms = (ManeuverState) message.getContent();
        assertEquals(expected, ms);
    }
    @SuperBuilder
    static class Bob extends BaseEntity {
        @Override
        public String getPath() {
            return "SYSTEM:Bob!";
        }
    }
}
```

A.492 SSTAF/src/testFeatures/military/mil.sstaftest.maneuver.entityagent/src/integrationTest/java/module-info.java

```
open module mil.sstaftest.maneuver.entityagent.integration {
    exports mil.sstaftest.maneuver.entityagent.integration;
    requires mil.sstaf.core;
    requires org.slf4j;
    requires mil.sstaf.blackboard.api;
    requires org.junit.jupiter.api;
    requires mil.sstaftest.maneuver.entityagent;
    requires mil.sstaftest.maneuver.api;
    uses mil.sstaf.core.features.Feature;
    uses mil.sstaf.blackboard.api.Blackboard;
    uses mil.sstaf.core.features.Agent;
    uses mil.sstaftest.maneuver.entityagent.ManeuverEntityAgent;
}
```

A.493 SSTAF/src/testFeatures/military/mil.sstaftest.maneuver.entityagent/src/integrationTest/resources/EmptyConfiguration.json

```
{
  "class": "mil.sstaf.core.configuration.SSTAFConfiguration",
  "moduleLayerDefinition": {
    "modules": [
    ],
    "modulePaths": [
    ]
  }
}
```

A.494 SSTAF/src/testFeatures/military/mil.sstaftest.maneuver.entityagent/src/integrationTest/resources/log4j2-test.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="INFO">
    <Appenders>
        <Console name="Console" target="SYSTEM_OUT">
            <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"/>
        </Console>
    </Appenders>
    <Loggers>
        <Root level="debug">
            <AppenderRef ref="Console"/>
        </Root>
    </Loggers>
</Configuration>
```

A.495 SSTAF/src/testFeatures/military/mil.sstaftest.maneuver.entityagent/src/main/java/mil/sstaftest/maneuver/entityagent/ManeuverEntityAgent.java

```
package mil.sstaftest.maneuver.entityagent;
import mil.sstaf.blackboard.api.Blackboard;
import mil.sstaf.core.entity.Address;
import mil.sstaf.core.entity.EntityEvent;
import mil.sstaf.core.entity.EntityHandle;
import mil.sstaf.core.entity.Message;
import mil.sstaf.core.features.BaseAgent;
import mil.sstaf.core.features.HandlerContent;
import mil.sstaf.core.features.ProcessingResult;
import mil.sstaf.core.features.Requires;
import mil.sstaf.core.util.SSTAFException;
import mil.sstaftest.maneuver.api.*;
import org.slf4j.LoggerFactory;
import org.slf4j.Logger;
import java.util.List;
import java.util.Optional;
/**
 * Agent that holds and updates the position, heading and speed of an Entity.
 */
public class ManeuverEntityAgent extends BaseAgent {
    public static final String FEATURE_NAME = "Maneuver Entity Agent";
    public static final int MAJOR_VERSION = 0;
    public static final int MINOR_VERSION = 1;
    public static final int PATCH_VERSION = 0;
    private EntityHandle centralAgentHandle = null;
    @Requires(name = "Blackboard", minorVersion = 1)
    private Blackboard blackboard;
    private static final Logger logger =
            LoggerFactory.getLogger(ManeuverEntityAgent.class);
    private long lastTimeStamp_ms = 0;
    private Heading heading = Heading.of(0.0);
    private Position position = Position.of(0.0, 0.0);
    private Speed speed = Speed.of(0.0);
    public ManeuverEntityAgent() {
        super(FEATURE_NAME, MAJOR_VERSION, MINOR_VERSION, PATCH_VERSION,
                false, "Planar maneuver agent");
    }

    private Position updatePosition(final Position position, final Heading heading,
                                    final Speed speed, double deltaT_ms) {
        double distance = speed.speed_ms * (deltaT_ms / 1000.0);
        double newXPos = position.x + distance * Math.cos(heading.heading_rads);
        double newYPos = position.y + distance * Math.sin(heading.heading_rads);
        return Position.of(newXPos, newYPos);
    }
    private ManeuverState updateManeuverState(final long currentTime_ms) {
        ManeuverState state = ManeuverState.of(ownerHandle, currentTime_ms, position, heading, sp
eed);
        blackboard.addEntry("ManeuverState", state, currentTime_ms);
        return state;
    }
    @Override
    public void init() {
        super.init();
    }
    @Override
    public ProcessingResult tick(long currentTime_ms) {
        logger.trace("{} ticking at {}", getInfoString(), currentTime_ms);
        double deltaT = currentTime_ms - lastTimeStamp_ms;
        lastTimeStamp_ms = currentTime_ms;
```

```
        if (position == null || heading == null) {
            throw new IllegalStateException("Initial Position and/or Heading are not set!");
        }
        position = updatePosition(position, heading, speed, deltaT);
        ManeuverState state = updateManeuverState(currentTime_ms);
        if (centralAgentHandle == null) {
            Optional<EntityHandle> optHandle = blackboard.getEntry("SYSTEM:EntityController", cur
rentTime_ms, EntityHandle.class);
            optHandle.ifPresentOrElse(
                    obj -> centralAgentHandle = obj,
                    () -> logger.warn("EntityController not set in ManeuverEntityAgent"));
        }
        if (centralAgentHandle == null) {
            logger.debug("{} centralAgent handle is null, returning empty ProcessingResult",
                    getInfoString());
            return ProcessingResult.empty();
        } else {
            var builder = EntityEvent.builder();
            builder.eventTime_ms(currentTime_ms);
            builder.source(Address.makeAddress(ownerHandle, getName()));
            builder.destination(Address.makeExternalAddress(centralAgentHandle));
            builder.content(state);
            EntityEvent event = builder.build();
            logger.trace("{} returning state {}", getInfoString(), state);
            return ProcessingResult.of(event);
        }
    }
    @Override
    public List<Class<? extends HandlerContent>> contentHandled() {
        return List.of(Heading.class, Position.class, Speed.class, ManeuverStateQuery.class,
                ManeuverState.class, ManeuverStateMap.class);
    }
    @Override
    public ProcessingResult process(HandlerContent arg, long scheduledTime_ms, long currentTime_m
s, Address from, long id, Address respondTo) {
        logger.debug("{} processing {} value = '{}'",
                getInfoString(), arg.getClass(), arg);
        boolean update = true;
        if (arg instanceof ManeuverState) {
            ManeuverState maneuverState = (ManeuverState) arg;
            logger.trace("{} updating with {}", getInfoString(), maneuverState);
            this.position = maneuverState.position;
            this.heading = maneuverState.heading;
            this.speed = maneuverState.speed;
            logger.trace("{} state = {} {} {}", getInfoString(),
                    this.position, this.heading, this.speed);
        } else if (arg instanceof Position) {
            this.position = (Position) arg;
            logger.trace("{} state = {} {} {}", getInfoString(),
                    this.position, this.heading, this.speed);
        } else if (arg instanceof Heading) {
            this.heading = (Heading) arg;
            logger.trace("{}, state = {} {} {}", getInfoString(),
                    this.position, this.heading, this.speed);
        } else if (arg instanceof Speed) {
            this.speed = (Speed) arg;
            logger.trace("{} state = {} {} {}", getInfoString(),
                    this.position, this.heading, this.speed);
        } else if (arg instanceof ManeuverStateQuery) {
            double deltaT = currentTime_ms - lastTimeStamp_ms;
            updatePosition(position, heading, speed, deltaT);
        } else if (arg instanceof ManeuverStateMap) {
            update = false;
            blackboard.addEntry("ManeuverStateMap", arg, currentTime_ms);
        } else {
            throw new SSTAFException("Can't process " + arg.getClass().getSimpleName());
        }
        if (update) {
```

```
            ManeuverState ms = updateManeuverState(currentTime_ms);
            logger.trace("{} returning current state {}", getInfoString(), ms);
            Message response = this.buildNormalResponse(ms, id, respondTo);
            return ProcessingResult.of(response);
        } else {
            return ProcessingResult.empty();
        }
    }
}
```

A.496 SSTAF/src/testFeatures/military/mil.sstaftest.maneuver.entityagent/src/main/java/module-info.java

```
import mil.sstaf.core.features.Agent;
import mil.sstaf.core.features.Feature;
import mil.sstaf.core.features.Handler;
import mil.sstaftest.maneuver.entityagent.ManeuverEntityAgent;
module mil.sstaftest.maneuver.entityagent {
    exports mil.sstaftest.maneuver.entityagent;
    requires mil.sstaf.core;
    requires mil.sstaftest.maneuver.api;
    requires mil.sstaf.blackboard.api;

    requires org.slf4j;
    provides Feature with ManeuverEntityAgent;
    provides Handler with ManeuverEntityAgent;
    provides Agent with ManeuverEntityAgent;
    opens mil.sstaftest.maneuver.entityagent
            to mil.sstaf.core;
}
```

A.497 SSTAF/src/testFeatures/military/mil.sstaftest.maneuver.entityagent/src/test/resources/log4j2-test.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="INFO">
    <Appenders>
        <Console name="Console" target="SYSTEM_OUT">
            <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"/>
        </Console>
    </Appenders>
    <Loggers>
        <Root level="debug">
            <AppenderRef ref="Console"/>
        </Root>
    </Loggers>
</Configuration>
```

A.498 SSTAF/src/testFeatures/support/mil.sstaftest.framework/build.gradle

```
plugins {
    id 'java-library'
}
ext.moduleName = 'mil.sstaftest.framework'
dependencies {
    implementation project(':framework:mil.sstaf.core')
    implementation project(':testFeatures:support:mil.sstaftest.mocks.api')
    implementation project(':testFeatures:support:mil.sstaftest.mocks.agent1')
    runtimeOnly project(':testFeatures:support:mil.sstaftest.mocks.handler1')
    runtimeOnly project(':testFeatures:support:mil.sstaftest.mocks.agent1')
    runtimeOnly project(':testFeatures:support:mil.sstaftest.mocks.brain')
    runtimeOnly project(':testFeatures:support:mil.sstaftest.mocks.pinky')
    testImplementation project(':verification:mil.sstaftest.util')
}
```

A.499 SSTAF/src/testFeatures/support/mil.sstaftest.framework/src/main/java/mil/sstaftest/framework/Dummy.java

```
package mil.sstaftest.framework;
/**
 * This class must be retained in order for the modules to build.
 */
public class Dummy {
}
```

A.500 SSTAF/src/testFeatures/support/mil.sstaftest.framework/src/main/java/module-info.java

```
module mil.sstaftest.framework {

    requires mil.sstaf.core;
    requires mil.sstaftest.mocks.api;
    requires mil.sstaftest.mocks.agentOne;
}
```

A.501 SSTAF/src/testFeatures/support/mil.sstaftest.framework/src/test/java/mil/sstaftest/entities/EntityFactoryTest.java

```
package mil.sstaftest.entities;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertDoesNotThrow;
class EntityFactoryTest {
    @Test
    void simpleTestWorks() {
        assertDoesNotThrow(() -> {
        });
    }
}
```

A.502 SSTAF/src/testFeatures/support/mil.sstaftest.framework/src/test/resources/entityTest.json

```
{
  "processors": {
    "handlers": [
      {
        "serviceName": "Handler1",
        "majorVersion": 3,
        "minorVersion": 1
      }
    ],
    "agents": [
      {
        "serviceName": "Agent1",
        "majorVersion": 3,
        "minorVersion": 1
      }
    ]
  },
  "providers": {
    "mil.sstaftest.mocks.api.BrainProvider": {
      "serviceName": "Brain"
    },
    "mil.sstaftest.mocks.api.PinkyProvider": {
      "serviceName": "Pinky"
    }
  }
}
```

A.503 SSTAF/src/testFeatures/support/mil.sstaftest.framework/src/test/resources/goodProvidersFile1.json

```
{
  "mil.sstaftest.mocks.api.PinkyProvider": {
    "serviceName": "Pinky"
  },
  "mil.sstaftest.mocks.api.BrainProvider": {
    "serviceName": "Brain"
  }
}
```

A.504 SSTAF/src/testFeatures/support/mil.sstaftest.framework/src/test/resources/handlers.json

```
{
  "handlers": [
    {
      "serviceName": "Handler1",
      "majorVersion": 3,
      "minorVersion": 1
    }
  ],
  "agents": [
    {
      "serviceName": "Agent1",
      "majorVersion": 3,
      "minorVersion": 1
    }
  ]
}
```

A.505 SSTAF/src/testFeatures/support/mil.sstaftest.mocks.agent1/build.gradle

```
plugins {
    id 'java-library'
}
ext.moduleName = 'mil.sstaftest.mocks.handler1'
dependencies {
    implementation project(':framework:mil.sstaf.core')
    implementation project(':testFeatures:support:mil.sstaftest.mocks.api')
    testImplementation project(':verification:mil.sstaftest.util')
}
```

A.506 SSTAF/src/testFeatures/support/mil.sstaftest.mocks.agent1/src/main/java/mil/sstaftest/mocks/agent1/Agent1.java

```
package mil.sstaftest.mocks.agent1;
import mil.sstaf.core.entity.Address;
import mil.sstaf.core.entity.Message;
import mil.sstaf.core.features.*;
import java.util.List;
public class Agent1 extends BaseAgent {
    public long lastTime_ms = 0;
    public int count = 0;
    public Agent1() {
        super("Agent1", 3, 1, 7, false, "");
    }
    @Override
    public ProcessingResult tick(long currentTime_ms) {
        lastTime_ms = currentTime_ms;
        count += 1;
        Message m = buildNormalResponse(IntContent.builder().intValue(count).build(),
                0, Address.makeExternalAddress(ownerHandle));
        return ProcessingResult.of(m);
    }
    @Override
    public List<Class<? extends HandlerContent>> contentHandled() {
        return List.of(LongContent.class, IntContent.class);
    }
    @Override
    public ProcessingResult process(HandlerContent arg, long scheduledTime_ms, long currentTime_m
s,
                                    Address from, long id, Address respondTo) {
        return ProcessingResult.empty();
    }
}
```

A.507 SSTAF/src/testFeatures/support/mil.sstaftest.mocks.agent1/src/main/java/module-info.java

```
import mil.sstaf.core.features.Agent;
import mil.sstaf.core.features.Feature;
import mil.sstaf.core.features.Handler;
import mil.sstaftest.mocks.agent1.Agent1;
module mil.sstaftest.mocks.agentOne {
    exports mil.sstaftest.mocks.agent1; // only because the test case digs into it.
    requires mil.sstaf.core;
    requires mil.sstaftest.mocks.api;

    provides Feature with Agent1;
    provides Handler with Agent1;
    provides Agent with Agent1;
    opens mil.sstaftest.mocks.agent1 to mil.sstaf.core;
}
```

A.508 SSTAF/src/testFeatures/support/mil.sstaftest.mocks.api/build.gradle

```
plugins {
    id 'java-library'
}
ext.moduleName = 'mil.sstaftest.mocks.api'
dependencies {
    implementation project(':framework:mil.sstaf.core')
    implementation project(':testFeatures:human:mil.sstaftest.anthropometry.api')
    testImplementation project(':verification:mil.sstaftest.util')
}
```

A.509 SSTAF/src/testFeatures/support/mil.sstaftest.mocks.api/src/main/java/mil/sstaftest/mocks/api/BrainProvider.java

```
package mil.sstaftest.mocks.api;
import mil.sstaf.core.features.Feature;
public interface BrainProvider extends Feature {
}
```

A.510 SSTAF/src/testFeatures/support/mil.sstaftest.mocks.api/src/main/java/mil/sstaftest/mocks/api/Command1.java

```
package mil.sstaftest.mocks.api;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import lombok.EqualsAndHashCode;
import lombok.experimental.SuperBuilder;
import lombok.extern.jackson.Jacksonized;
import mil.sstaf.core.features.HandlerContent;
@SuperBuilder
@Jacksonized
@JsonTypeInfo(use = JsonTypeInfo.Id.CLASS, property = "class")
@EqualsAndHashCode(callSuper = true)
public class Command1 extends HandlerContent {
    public String string;
    public boolean done;
    public Command1() {
        super();
        string = "Waiting";
        done = false;
    }
}
```

A.511 SSTAF/src/testFeatures/support/mil.sstaftest.mocks.api/src/main/java/mil/sstaftest/mocks/api/GetAnthroMsg.java

```
package mil.sstaftest.mocks.api;
import mil.sstaftest.anthropometry.api.Anthropometry;
public class GetAnthroMsg {
    public Anthropometry anthropometry;
}
```

A.512 SSTAF/src/testFeatures/support/mil.sstaftest.mocks.api/src/main/java/mil/sstaftest/mocks/api/PinkyProvider.java

```
package mil.sstaftest.mocks.api;
import mil.sstaf.core.features.Feature;
public interface PinkyProvider extends Feature {
}
```

A.513 SSTAF/src/testFeatures/support/mil.sstaftest.mocks.api/src/main/java/mil/sstaftest/mocks/api/SnowballProvider.java

```
package mil.sstaftest.mocks.api;
import mil.sstaf.core.features.Feature;
public interface SnowballProvider extends Feature {
}
```

A.514 SSTAF/src/testFeatures/support/mil.sstaftest.mocks.api/src/main/java/module-info.java

```
module mil.sstaftest.mocks.api {
    exports mil.sstaftest.mocks.api;
    requires mil.sstaf.core;
    requires mil.sstaftest.anthropometry.api;
}
```

A.515 SSTAF/src/testFeatures/support/mil.sstaftest.mocks.brain/build.gradle

```
plugins {
    id 'java-library'
}
ext.moduleName = 'mil.sstaftest.mocks.brain'
dependencies {
    implementation project(':framework:mil.sstaf.core')
    implementation project(':testFeatures:support:mil.sstaftest.mocks.api')
    testImplementation project(':verification:mil.sstaftest.util')
}
```

A.516 SSTAF/src/testFeatures/support/mil.sstaftest.mocks.brain/src/main/java/mil/sstaftest/mocks/brain/Brain.java

```
package mil.sstaftest.mocks.brain;
import mil.sstaf.core.features.BaseFeature;
import mil.sstaftest.mocks.api.BrainProvider;
public class Brain extends BaseFeature implements BrainProvider {
    public Brain() {
        super("Brain", 314, 0, 0, false, "");
    }
}
```

A.517 SSTAF/src/testFeatures/support/mil.sstaftest.mocks.brain/src/main/java/module-info.java

```
import mil.sstaf.core.features.Feature;
import mil.sstaftest.mocks.api.BrainProvider;
import mil.sstaftest.mocks.brain.Brain;
module mil.sstaftest.mocks.brain {
    exports mil.sstaftest.mocks.brain; // only because the test case digs into it.
    requires mil.sstaf.core;
    requires mil.sstaftest.mocks.api;

    provides BrainProvider with Brain;
    provides Feature with Brain;
    opens mil.sstaftest.mocks.brain to mil.sstaf.core;
}
```

A.518 SSTAF/src/testFeatures/support/mil.sstaftest.mocks.handler1/build.gradle

```
plugins {
    id 'java-library'
}
ext.moduleName = 'mil.sstaftest.mocks.handler1'
dependencies {
    implementation project(':framework:mil.sstaf.core')
    implementation project(':testFeatures:support:mil.sstaftest.mocks.api')
    testImplementation project(':verification:mil.sstaftest.util')
}
```

A.519 SSTAF/src/testFeatures/support/mil.sstaftest.mocks.handler1/src/main/java/mil/sstaftest/mocks/handler1/Handler1.java

```
package mil.sstaftest.mocks.handler1;
import mil.sstaf.core.entity.Address;
import mil.sstaf.core.entity.Message;
import mil.sstaf.core.features.*;
import mil.sstaf.core.util.SSTAFException;
import mil.sstaftest.mocks.api.Command1;
import mil.sstaftest.mocks.api.PinkyProvider;
import java.util.List;
public class Handler1 extends BaseHandler {
    @Requires(name = "Pinky", minorVersion = 1)
    public PinkyProvider pinky = null;
    int x = 3;
    String y = "xxxxx";
    Object q = new Object();
    public Handler1() {
        super("Handler1", 3, 1, 7, false, "Handle it Roy. Handle it, handle it");
    }
    @Override
    public List<Class<? extends HandlerContent>> contentHandled() {
        return List.of(Command1.class, StringContent.class);
    }
    @Override
    public String toString() {
        return "Handler1{" +
                "x=" + x +
                ", y='" + y + '\'' +
                ", q=" + q +
                ", pinky=" + pinky +
                '}';
    }
    @Override
    public void init() {
        super.init();
        if (pinky == null) {
            throw new SSTAFException("Narf!");
        }
    }
    @Override
    public ProcessingResult process(HandlerContent arg, long scheduledTime_ms, long currentTime_m
s,
                                    Address from, long id, Address respondTo) {
        if (arg instanceof Command1) {
            Command1 tc1 = (Command1) arg;
            tc1.string = "Got it";
            tc1.done = true;
            Message m = this.buildNormalResponse(tc1, id, respondTo);
            return ProcessingResult.of(m);
        } else {
            return ProcessingResult.empty();
        }
    }
}
```

A.520 SSTAF/src/testFeatures/support/mil.sstaftest.mocks.handler1/src/main/java/module-info.java

```
import mil.sstaf.core.features.Feature;
import mil.sstaf.core.features.Handler;
import mil.sstaftest.mocks.handler1.Handler1;
module mil.sstaftest.mocks.handlerOne {
    exports mil.sstaftest.mocks.handler1; // only because the test case digs into it.
    requires mil.sstaf.core;
    requires mil.sstaftest.mocks.api;

    provides Feature with Handler1;
    provides Handler with Handler1;
    opens mil.sstaftest.mocks.handler1 to mil.sstaf.core;
}
```

A.521 SSTAF/src/testFeatures/support/mil.sstaftest.mocks.pinky/build.gradle

```
plugins {
    id 'java-library'
}
ext.moduleName = 'mil.sstaftest.mocks.pinky'
dependencies {
    implementation project(':framework:mil.sstaf.core')
    implementation project(':testFeatures:support:mil.sstaftest.mocks.api')
}
task copyJar(type: Copy) {
    from jar
    into "../../../../src/framework/mil.sstaf.core/src/integrationTest/resources/modules/pinky"
}
build.dependsOn copyJar
```

A.522 SSTAF/src/testFeatures/support/mil.sstaftest.mocks.pinky/src/main/java/mil/sstaftest/mocks/pinky/Pinky.java

```
package mil.sstaftest.mocks.pinky;
import mil.sstaf.core.entity.Address;
import mil.sstaf.core.features.BaseHandler;
import mil.sstaf.core.features.HandlerContent;
import mil.sstaf.core.features.ProcessingResult;
import mil.sstaf.core.features.StringContent;
import mil.sstaftest.mocks.api.PinkyProvider;
import java.util.List;
public class Pinky extends BaseHandler implements PinkyProvider {
    public Pinky() {
        super("Pinky", 13, 0, 0, false, "");
    }
    @Override
    public List<Class<? extends HandlerContent>> contentHandled() {
        return List.of(StringContent.class);
    }
    @Override
    public ProcessingResult process(HandlerContent arg, long scheduledTime_ms, long currentTime_m
s, Address from, long id, Address respondTo) {
        return null;
    }
}
```

A.523 SSTAF/src/testFeatures/support/mil.sstaftest.mocks.pinky/src/main/java/module-info.java

```
import mil.sstaf.core.features.Feature;
import mil.sstaftest.mocks.api.PinkyProvider;
import mil.sstaftest.mocks.pinky.Pinky;
module mil.sstaftest.mocks.pinky {
    exports mil.sstaftest.mocks.pinky; // only because the test case digs into it.
    requires mil.sstaf.core;
    requires mil.sstaftest.mocks.api;

    provides Feature with Pinky;
    provides PinkyProvider with Pinky;
    opens mil.sstaftest.mocks.pinky to mil.sstaf.core;
}
```

A.524 SSTAF/src/testModules/mil.sstaftest.barney/build.gradle

```
plugins {
    id 'java-library'
}
task copyJar(type: Copy) {
    from jar
    into "../../framework/mil.sstaf.core/src/test/resources/moduleLayers/dir2"
}
build.dependsOn copyJar
```

A.525 SSTAF/src/testModules/mil.sstaftest.barney/src/main/java/mil/sstaftest/barney/BarneyImpl.java

```
package mil.sstaftest.barney;
import java.util.function.Predicate;
public class BarneyImpl implements Predicate<String> {
    public BarneyImpl() {
    }
    @Override
    public boolean test(String s) {
        return s != null && s.length() % 2 == System.currentTimeMillis() % 2;
    }
}
```

A.526 SSTAF/src/testModules/mil.sstaftest.barney/src/main/java/module-info.java

```
import mil.sstaftest.barney.BarneyImpl;
import java.util.function.Predicate;
open module mil.sstaftest.barney {
    exports mil.sstaftest.barney;
    provides Predicate with BarneyImpl;
}
```

A.527 SSTAF/src/testModules/mil.sstaftest.betty/build.gradle

```
plugins {
    id 'java-library'
}
task copyJar(type: Copy) {
    from jar
    into "../../framework/mil.sstaf.core/src/test/resources/moduleLayers/dir4"
}
build.dependsOn copyJar
```

A.528 SSTAF/src/testModules/mil.sstaftest.betty/src/main/java/mil/sstaftest/betty/BettyImpl.java

```
package mil.sstaftest.betty;
import java.util.function.Predicate;
public class BettyImpl implements Predicate<String> {
    public BettyImpl() {
    }
    @Override
    public boolean test(String s) {
        return s != null && s.length() % 2 == System.currentTimeMillis() % 2;
    }
}
```

A.529 SSTAF/src/testModules/mil.sstaftest.betty/src/main/java/module-info.java

```
import mil.sstaftest.betty.BettyImpl;
import java.util.function.Predicate;
open module mil.sstaftest.betty {
    exports mil.sstaftest.betty;
    provides Predicate with BettyImpl;
}
```

A.530 SSTAF/src/testModules/mil.sstaftest.fred/build.gradle

```
plugins {
    id 'java-library'
}
task copyJar(type: Copy) {
    from jar
    into "../../framework/mil.sstaf.core/src/test/resources/moduleLayers/dir1"
}
build.dependsOn copyJar
```

A.531 SSTAF/src/testModules/mil.sstaftest.fred/src/main/java/mil/sstaftest/fred/FredImpl.java

```
package mil.sstaftest.fred;
import java.util.function.Predicate;
public class FredImpl implements Predicate<String> {
    public FredImpl() {
    }
    @Override
    public boolean test(String s) {
        return s != null && s.length() % 2 == System.currentTimeMillis() % 2;
    }
}
```

A.532 SSTAF/src/testModules/mil.sstaftest.fred/src/main/java/module-info.java

```
import mil.sstaftest.fred.FredImpl;
import java.util.function.Predicate;
open module mil.sstaftest.fred {
    exports mil.sstaftest.fred;
    provides Predicate with FredImpl;
}
```

A.533 SSTAF/src/testModules/mil.sstaftest.wilma/build.gradle

```
plugins {
    id 'java-library'
}
task copyJar(type: Copy) {
    from jar
    into "../../framework/mil.sstaf.core/src/test/resources/moduleLayers/dir3"
}
build.dependsOn copyJar
```

A.534 SSTAF/src/testModules/mil.sstaftest.wilma/src/main/java/mil/sstaftest/wilma/WilmaImpl.java

```
package mil.sstaftest.wilma;
import java.util.function.Predicate;
public class WilmaImpl implements Predicate<String> {
    public WilmaImpl() {
    }
    @Override
    public boolean test(String s) {
        return s != null && s.length() % 2 == System.currentTimeMillis() % 2;
    }
}
```

A.535 SSTAF/src/testModules/mil.sstaftest.wilma/src/main/java/module-info.java

```
import mil.sstaftest.wilma.WilmaImpl;
import java.util.function.Predicate;
open module mil.sstaftest.wilma {
    exports mil.sstaftest.wilma;
    provides Predicate with WilmaImpl;
}
```

A.536 SSTAF/src/verification/mil.sstaftest.util/build.gradle

```
plugins {
    id 'java-library'
    id 'sstaf'
}
ext.moduleName = 'mil.sstaftest.util'
sourceSets {
    main {
        resources {
            srcDirs "src/main/resources", "src/main/python"
        }
    }
    test {
        resources {
            srcDirs "src/test/resources", "src/main/python", "src/test/python"
        }
    }
}
jar {
    manifest {
        def fileList = []
        ['src/test/python', 'src/test/resource'].each { String dir ->
            fileTree(dir) {
                include '**/*.py'
                include '**/*.json'
            }.collect {
                println(it.getPath())
                relativePath(it)
            }.each {
                println(it)
                def trimmed = it.replace("$dir/",'')
                fileList << trimmed
                println(fileList)
            }
        }
        attributes('SSTAF-Module' : 'mil.sstaf.core')
        attributes('SSTAF-Resources-To-Extract' : fileList.join(' '))
    }
}
dependencies {
    implementation project(':framework:mil.sstaf.core')
    implementation 'org.junit.jupiter:junit-jupiter:5.7.2'
    implementation group: 'org.slf4j', name: 'slf4j-api', version: '1.7.32'
}
```

A.537 SSTAF/src/verification/mil.sstaftest.util/src/main/java/mil/sstaftest/util/BaseAgentTest.java

```
package mil.sstaftest.util;
import mil.sstaf.core.features.Agent;
import mil.sstaf.core.features.ProcessingResult;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Nested;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertNotNull;
/**
 * Base class for testing implementations of {@link Agent}
 * <p>
 * This class extends {@link BaseHandlerTest} to provide a framework and series of tests to confi
rm than
 * an implementation of the {@code Agent} interface fulfils all of the contractual obligations ne
cessary
 * for the implementation to be used in SSTAF.
 * </p>
 */
public abstract class BaseAgentTest<T extends Agent> extends BaseHandlerTest<T> {
    @Nested
    @DisplayName("Check requirements for implementations of 'Agent'")
    public class AgentContractTests {
        /**
         * Confirms that the {@code tick()} method works and that it returns a valid {@link Proce
ssingResult}.
         */
        @Test
        @DisplayName("Confirm that tick() returns a ProcessingResult")
        public void checkMultipleTicks() {
            for (long time = -10000; time < 100000L; time += 1000) {
                tickCheck(time);
            }
        }
        private void tickCheck(long time) {
            T agent = setupFeature();
            ProcessingResult pr = agent.tick(time);
            assertNotNull(pr, agent.getClass().getName()
                    + ".tick(" + time + ") returned a null ProcessingResult");
        }

    }
}
```

A.538 SSTAF/src/verification/mil.sstaftest.util/src/main/java/mil/sstaftest/util/BaseFeatureIntegrationTest.java

```
package mil.sstaftest.util;
import mil.sstaf.core.entity.EntityHandle;
import mil.sstaf.core.features.*;
import mil.sstaf.core.util.Injector;
import org.apache.commons.math3.random.MersenneTwister;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Nested;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.condition.DisabledIfSystemProperty;
import java.util.HashMap;
import java.util.Map;
import java.util.function.Supplier;
import static org.junit.jupiter.api.Assertions.*;
/**
 * Base class for integration testing implementations of {@link Feature}.
 * <p>
 * This class implements a basic integration test for a Feature
 * <p>
 * Using this test framework requires some simple ceremony.
 * </p>
 * <ol>
 *     <li>The implementation classes must pass the {@code FeatureSpecification} for the feature
to be
 *     tested in to this constructor. Typically this will look like:
 *      <pre>
 *      {@code super(FeatureSpecification.builder() ... .build()}
 *      </pre>
 *      The specification should include the interface class for the feature.
 *     </li>
 *
 * <li>
 *     The configuration for each {@code Feature} must be defined. This is accomplished in the {@
code static}
 *     block using the {@link #setDefaultConfiguration(String, FeatureConfiguration)} method.
 * </li>
 * </ol>
 *
 * @param <T> The type of the {@code Feature} to be tested.
 * @author Ron Bowers
 * @version 1.0
 * @since 1.0.0
 */
@DisabledIfSystemProperty(named = "sstaf.disableBaseFeatureTests", matches = "true")
abstract public class BaseFeatureIntegrationTest<T extends Feature,
        S extends FeatureConfiguration> {
    protected static Map<String, FeatureConfiguration> defaultConfigMap = new HashMap<>();
    protected Map<FeatureSpecification, Feature> featureMap = new HashMap<>();
    protected final FeatureSpecification featureSpecification;
    protected Class<T> apiClass;
    protected T feature;
    protected Map<String, FeatureConfiguration> configurationMap;
    protected MersenneTwister randomGenerator = new MersenneTwister(12345);
    /**
     * Constructor
     */
    protected BaseFeatureIntegrationTest(Class<T> apiClass, FeatureSpecification fs) {
        this.apiClass = apiClass;
        this.featureSpecification = fs;
        this.configurationMap = defaultConfigMap;
    }
    /**
```

```
     * Sets the configuration for the {@code Feature} under test as well as any other features, h
andlers or agents that
     * are required by this {@code Feature}.
     *
     * @param featureName the name of the {@code Feature} to which the configuration is to be app
lied
     * @param cfg         the configuration object
     */
    protected static void setDefaultConfiguration(String featureName, FeatureConfiguration cfg) {
        defaultConfigMap.put(featureName, cfg);
    }

    /**
     * Provides the next random seed from the random number generator.
     *
     * @return a new random long
     */
    protected long getRandomSeed() {
        return randomGenerator.nextLong();
    }
    /**
     * @return the configuration for the {@code Feature
     */
    protected FeatureConfiguration getFeatureConfiguration(Feature f) {
        assertNotNull(f, "Feature is null");
        String name = f.getName();
        assertNotNull(name, "Feature getName() returned null");
        FeatureConfiguration featureConfig;
        if (configurationMap.containsKey(name)) {
            featureConfig = configurationMap.get(name);
        } else {
            featureConfig = FeatureConfiguration.builder().build();
        }
        return featureConfig;
    }
    /**
     * Configures the {@code Feature} under test for use given the default configurations.
     *
     * @return the {@code Feature}
     */
    public T loadAndResolveFeature() {
        Resolver resolver = new Resolver(featureMap, configurationMap,
                EntityHandle.makeDummyHandle(), getRandomSeed(), ModuleLayer.boot());
        Feature f = resolver.loadAndResolveDependencies(featureSpecification);
        if (apiClass.isAssignableFrom(f.getClass())) {
            T myFeature = apiClass.cast(f);assertNotNull(myFeature, "Feature is null");
            resolveFeature(myFeature);
            configureFeature(myFeature);
            myFeature.init();
            return myFeature;
        } else {
            String report = Loaders.generateServiceReport(Feature.class);
            fail("Could not find service!\n" + report);
            return null;
        }
    }
    /**
     * Configures the {@code Feature} using the default configuration.
     *
     * @param myFeature the {@code Feature}
     */
    protected void configureFeature(T myFeature) {
        FeatureConfiguration configuration = getFeatureConfiguration(myFeature);
        configuration.setSeed(getRandomSeed());
        myFeature.configure(configuration);
    }
    /**
```

```
     * Resolves any dependencies expressed by the {@code Feature}. The default configuration set
will be used
     * to configure any newly-loaded features, handlers or agents.
     *
     * @param myFeature the {@code Feature}
     */
    protected void resolveFeature(T myFeature) {
        Resolver resolver = Resolver.makeTransientResolver(configurationMap);
        resolver.resolveDependencies(myFeature);
        EntityHandle owner = EntityHandle.makeDummyHandle();
        Injector.inject(myFeature, owner);
    }
    /**
     * Tests
     */
    @Nested
    @DisplayName("Check service loading and resolution")
    public class FeatureContractTests {
        @Test
        @DisplayName("Confirm that the module containing this feature can be loaded and transitiv
e dependencies resolved")
        public void testLoadAndResolve() {
            T myFeature = loadAndResolveFeature();
            assertNotNull(myFeature);
            assertEquals(featureSpecification.featureName, myFeature.getName());
            assertTrue(myFeature.getMajorVersion() >= featureSpecification.majorVersion);
        }
    }
}
```

A.539 SSTAF/src/verification/mil.sstaftest.util/src/main/java/mil/sstaftest/util/BaseFeatureTest.java

```
package mil.sstaftest.util;
import mil.sstaf.core.entity.EntityHandle;
import mil.sstaf.core.features.Feature;
import mil.sstaf.core.features.FeatureConfiguration;
import mil.sstaf.core.features.Loaders;
import mil.sstaf.core.features.Resolver;
import mil.sstaf.core.util.Injector;
import org.apache.commons.math3.random.MersenneTwister;
import org.junit.jupiter.api.*;
import org.junit.jupiter.api.condition.DisabledIfSystemProperty;
import java.lang.reflect.Constructor;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.ServiceLoader;
import static org.junit.jupiter.api.Assertions.*;
/**
 * Base class for testing implementations of {@link Feature}.
 * <p>
 * This class implements a framework and series of tests to confirm that an implementation of the
{@code Feature}
 * interface fulfils all of the contractual obligations necessary for the implementation to be us
ed in SSTAF.
 * At least one test class for the implementation should extend this class. If multiple test clas
ses extends this
 * class, the base tests can be skipped by setting the {@code sstaf.disableFeatureBaseTests} syst
em property to
 * <em>true</em> in the {@code static} block at the beginning of those test classes.
 * <p>
 * Using this test framework requires some simple ceremony.
 * </p>
 * <ol>
 * <li>
 *       The concrete test class must implement the {@link #buildFeature()} method that returns a
n instance
 *       of the {@code Feature} under test.
 * </li>
 *
 * <li>
 *     The feature classes that are required by the {@code Feature} under test and must be dynami
cally loaded
 *     must be specified. This is done in a {@code static} block with this construct:
 *     <pre>
 *     {@code preloadedClasses = List.of("class1Name", "class2Name", ...); }
 *     </pre>
 * </li>
 *
 * <li>
 *     The configuration for each {@code Feature} must be defined. This is accomplished in the {@
code static}
 *     block using the {@link #setDefaultConfiguration(String, FeatureConfiguration)} method.
 * </li>
 * </ol>
 *
 * @param <T> The type of the {@code Feature} to be tested.
 * @author Ron Bowers
 * @version 1.0
 * @since 1.0.0
 */
abstract public class BaseFeatureTest<T extends Feature> {
    protected static List<String> preloadedClasses;
    protected static Map<String, FeatureConfiguration> defaultConfigMap = new HashMap<>();
```

```
    protected T feature;
    protected Map<String, FeatureConfiguration> configurationMap;
    protected MersenneTwister randomGenerator = new MersenneTwister(12345);
    /**
     * Constructor
     */
    protected BaseFeatureTest() {
       this.configurationMap = new HashMap<>(defaultConfigMap);
    }
    /**
     * Sets the configuration for the {@code Feature} under test as well as any other features, h
andlers or agents that
     * are required by this {@code Feature}.
     *
     * @param featureName the name of the {@code Feature} to which the configuration is to be app
lied
     * @param cfg         the configuration object
     */
    protected static void setDefaultConfiguration(String featureName, FeatureConfiguration cfg) {
        defaultConfigMap.put(featureName, cfg);
    }
    /**
     * Loads the required classes
     */
    @BeforeEach
    protected void preload() {
        System.setProperty("sstaf.preloadFeatureClasses", "true");
        try {
            Loaders.preloadFeatureClasses(preloadedClasses);
        } catch (ClassNotFoundException e) {
            fail("A required class was not found during preloading.\n" + e.getMessage());
        }
    }
    /**
     * Builds and returns the {@code Feature} to be tested
     *
     * @return the {@code Feature} instance
     */
    protected abstract T buildFeature();
    /**
     * Provides the next random seed from the random number generator.
     *
     * @return a new random long
     */
    protected long getRandomSeed() {
        return randomGenerator.nextLong();
    }
    /**
     * @return the configuration for the {@code Feature
     */
    protected FeatureConfiguration getFeatureConfiguration(Feature f) {
        assertNotNull(f, "Feature is null");
        String name = f.getName();
        assertNotNull(name, "Feature getName() returned null");
        FeatureConfiguration featureConfig;
        if (configurationMap.containsKey(name)) {
            featureConfig = configurationMap.get(f.getName());
        } else {
            featureConfig = FeatureConfiguration.builder().build();
        }
        return featureConfig;
    }
    /**
     * Configures the {@code Feature} under test for use given the default configurations.
     *
     * @return the {@code Feature}
     */
    public T setupFeature() {
```

```
        T myFeature = buildFeature();
        assertNotNull(myFeature, "Feature is null");
        resolveFeature(myFeature);
        configureFeature(myFeature);
        myFeature.init();
        return myFeature;
    }
    /**
     * Configures the {@code Feature} using the default configuration.
     *
     * @param myFeature the {@code Feature}
     */
    protected void configureFeature(T myFeature) {
        FeatureConfiguration configuration = getFeatureConfiguration(myFeature);
        configuration.setSeed(getRandomSeed());
        myFeature.configure(configuration);
    }
    /**
     * Resolves any dependencies expressed by the {@code Feature}. The default configuration set
will be used
     * to configure any newly-loaded features, handlers or agents.
     *
     * @param myFeature the {@code Feature}
     */
    protected void resolveFeature(T myFeature) {
        Resolver resolver = Resolver.makeTransientResolver(configurationMap);
        resolver.resolveDependencies(myFeature);
        EntityHandle owner = EntityHandle.makeDummyHandle();
        Injector.inject(myFeature, owner);
    }
    /**
     * Tests
     */
    @Nested
    @DisabledIfSystemProperty(named = "sstaf.disableFeatureBaseTests", matches = "true")
    @DisplayName("Check requirements for implementations of 'Feature'")
    public class FeatureContractTests {
        /**
         * Confirms that the {@code Feature} under test has a no-arg constructor. A no-arg
         * constructor is required by the {@link ServiceLoader} mechanism.
         */
        @Test
        @DisplayName("Confirm that the Feature has a no-arg constructor")
        public void checkNoArg() {
            T feature = buildFeature();
            boolean foundOne = false;
            for (Constructor<?> constructor : feature.getClass().getConstructors()) {
                if (constructor.getParameterCount() == 0) {
                    foundOne = true;
                    break;
                }
            }
            Assertions.assertTrue(foundOne, "Did not find a no-arg constructor in "
                    + feature.getClass().getName());
        }
        /**
         * Confirms that the {@link EntityHandle} associated with the owner of this {@code Featur
e} is retrievable
         * and correct.
         */
        @Test
        @DisplayName("Confirm that getOwner() returns the injected owner handle")
        public void getOwnerWorks() {
            T myFeature = buildFeature();
            //
            // Inline this so we have access to the dummy owner.
            //
            Resolver resolver = Resolver.makeTransientResolver(configurationMap);
```

```
            resolver.resolveDependencies(myFeature);
            EntityHandle dummy = EntityHandle.makeDummyHandle();
            Injector.inject(myFeature, dummy);
            configureFeature(myFeature);
            final String methodName = myFeature.getClass().getSimpleName() + ".getOwner()";
            EntityHandle eh = myFeature.getOwner();
            assertNotNull(eh, methodName + " returned null");
            Assertions.assertEquals(dummy, myFeature.getOwner(), methodName + " did not return th
e expected owner");
        }
        /**
         * Confirms tha the {@code configure()} and {@code isConfigured()} mechanisms work correc
tly
         */
        @Test
        @DisplayName("Confirm calling configure() works and sets isConfigured() to true")
        public void configureSetsIsConfigured() {
            T myFeature = buildFeature();
            resolveFeature(myFeature);
            final String methodName1 = myFeature.getClass().getSimpleName() + ".configure()";
            Assertions.assertDoesNotThrow(() -> configureFeature(myFeature), methodName1 + " thre
w");
            final String methodName2 = myFeature.getClass().getSimpleName() + ".isConfigured()";
            Assertions.assertTrue(myFeature.isConfigured(), methodName2 + " did not return 'true'
as expected.");
        }
        /**
         * Confirms tha the {@code init()} and {@code isInitialized()} mechanisms work correctly
         */
        @Test
        @DisplayName("Confirm calling init() works and sets isInitialized() to true")
        public void initWorks() {
            T myFeature = buildFeature();
            resolveFeature(myFeature);
            configureFeature(myFeature);
            final String methodName1 = myFeature.getClass().getSimpleName() + ".init()";
            Assertions.assertDoesNotThrow(myFeature::init, methodName1 + " threw an exception");
            final String methodName2 = myFeature.getClass().getSimpleName() + ".isInitialized()";
            Assertions.assertTrue(myFeature.isConfigured(), methodName2 + " did not return 'true'
as expected.");
        }
        /**
         * Confirms that the name, version number and description values are reasonable.
         * <p>
         * Note that the name and version number are decoupled from the class name and the artifa
ct version number
         * generated by the build.
         * +9
         */
        @Test
        @DisplayName("Confirm name, description and version info are reasonable")
        public void checkSpecInfo() {
            T myFeature = setupFeature();
            String className = myFeature.getClass().getSimpleName();
            String name = myFeature.getName();
            String description = myFeature.getDescription();
            int majorVersion = myFeature.getMajorVersion();
            int minorVersion = myFeature.getMinorVersion();
            int patchVersion = myFeature.getPatchVersion();
            assertNotNull(name, className + ".getName() returned null. It must the 'official' use
r-comprehensible name.");
            assertNotNull(description, className + ".getDescription returned null. It should retu
rn something helpful");
            assertFalse(name.isEmpty(), className + ".getName() returned an empty String. It must
the 'official' user-comprehensible name.");
            assertFalse(description.isEmpty(), className + ".getDescription() returned an empty S
tring. It must the 'official' user-comprehensible name.");
```

```
            assertTrue(majorVersion >= 0, className + ".getMajorVersion() returned < 0, should be
>= 0");
            assertTrue(minorVersion >= 0, className + ".getMinorVersion() returned < 0, should be
>= 0");
            assertTrue(patchVersion >= 0, className + ".getPatchVersion() returned < 0, should be
>= 0");
            assertFalse(majorVersion == 0 && minorVersion == 0 && patchVersion == 0,
                    "In " + className + " all version values are 0, at least one must be non-zero
");
        }
    }
}
```

A.540 SSTAF/src/verification/mil.sstaftest.util/src/main/java/mil/sstaftest/util/BaseHandlerTest.java

```
package mil.sstaftest.util;
import mil.sstaf.core.entity.Address;
import mil.sstaf.core.entity.ErrorResponse;
import mil.sstaf.core.features.Agent;
import mil.sstaf.core.features.Handler;
import mil.sstaf.core.features.HandlerContent;
import mil.sstaf.core.features.ProcessingResult;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Nested;
import org.junit.jupiter.api.Test;
import java.util.ArrayList;
import java.util.List;
import static org.junit.jupiter.api.Assertions.*;
/**
 * Base class for testing implementations of {@link Handler}
 * <p>
 * This class extends {@link BaseFeatureTest} to provide a frameowkr and series of tests to confi
rm than
 * an implementation of the {@code Handler} interface fulfils all of the contractual obligations
necessary
 * for the implementation to be used in SSTAF.
 * </p>
 *
 * <p>Like {@code BaseFeatureTest}, using this test class requires some setup</p>
 *
 * <ul>
 *     <li>
 *     The concrete test class must create a valid instance of each supported message content cla
ss
 *     and add it to the {@cod sampleMessages} list. This is done in a {@code static} block with
 *     this construct:
 *     <pre><code>
 *     var message1 = new MessageOne()
 *     var message2 = new MessageTwo()
 *     ...
 *     sampleMessages = List.of(message1, message2, ...);
 *     </code></pre>
 *     </li>*
 * </ul>
 *
 * @param <T>
 */
abstract public class BaseHandlerTest<T extends Handler> extends BaseFeatureTest<T> {
    protected static List<? extends HandlerContent> sampleMessages;
    static class NonsenseMessage {
    }
    /**
     * Tests
     */
    @Nested
    @DisplayName("Check requirements for implementations of 'Handler'")
    public class HandlerContractTests {
        /**
         * Confirms that the {@code contentHandled()} method works.
         * <p>
         * Note that if the implementation is an {@code Agent}, it is not required to handle mess
ages. This
         * is because an {@code Agent} will be activated by the {@code tick()} method.
         */
        @Test
        @DisplayName("Confirm that contentHandled() returns non-null and non-empty")
        public void checkContentHandled() {
```

```
            T handler = setupFeature();
            List<Class<? extends HandlerContent>> ch = handler.contentHandled();
            String className = handler.getClass().getSimpleName();
            assertNotNull(ch, className + ".contentHandled() returned null, it must return a vali
d List");
            if (!(handler instanceof Agent)) {
                assertFalse(ch.isEmpty(), className + ".contentHandled() returned an empty list.
Handlers must 'handle' at least one message type");
            }
        }
        /**
         * Confirms that the {@code getAddress()} mechanism works.
         */
        @Test
        @DisplayName("Confirm that getAddress() provides the expected Address")
        public void checkAddress() {
            T handler = setupFeature();
            Address a = handler.getAddress();
            String className = handler.getClass().getSimpleName();
            assertNotNull(a, className + ".getAddress() returned null. It should return a valid A
ddress object");
            assertNotNull(a.entityHandle, "The Address returned from " + className + ".getAddress
() has a null EntityHandle");
            assertEquals(handler.getName(), a.handlerName,
                    "The Address returned from " + className + ".getAddress() contained the wrong
Handler name");
        }
        /**
         * Confirms that the {@code sampleMessage} list is configured. This is a precondition to
other tests.
         */
        @Test
        @DisplayName("Confirm that the sample message list is populated")
        public void checkMessageList() {
            T handler = setupFeature();
            if (handler.contentHandled().size() > 0) {
                assertNotNull(sampleMessages, "The sampleMessages list is null, must be a valid L
ist");
                List<Class<? extends HandlerContent>> ch = handler.contentHandled();
                assertEquals(ch.size(), sampleMessages.size(),
                        "The contentHandled() list and sampleMessages list are different sizes");
                List<Class<?>> lsmc = new ArrayList<>();
                for (Object o : sampleMessages) {
                    Class<?> c = o.getClass();
                    lsmc.add(c);
                    assertTrue(ch.contains(c),
                            "SampleMessage class " + c.getName() + " is not in content handled li
st");
                }
                for (Class<?> o : ch) {
                    assertTrue(lsmc.contains(o),
                            "Handled class " + o.getName() + " is not represented in the sample m
essages list");
                }
            }
        }
        /**
         * Confirms that each of the sample messages can be processed.
         */
        @Test
        @DisplayName("Confirm that processing the sample messages works")
        public void checkProcess() {
            T handler = setupFeature();
            if (handler.contentHandled().size() > 0) {
                for (HandlerContent o : sampleMessages) {
                    ProcessingResult pr = handler.process(o, 1000, 1000, Address.NOWHERE, 1, Addr
ess.NOWHERE);
                    assertNotNull(pr, handler.getClass().getName()
```

```
                            + ".process(" + o.getClass().getSimpleName() + ") returned a null Pro
cessingResult");
                }
            }
        }
        class Garbage extends HandlerContent {
        }
        /**
         * Confirms that if an unsupported message is passed to the {@code Handler}, an {@code Er
rorResponse} is
         * generated.
         */
        @Test
        @DisplayName("Confirm that processing an unsupported results in a ProcessingResult with a
single error message")
        public void checkUnsupported() {
            T handler = setupFeature();
            ProcessingResult pr = handler.process(new Garbage(), 1000, 1000, Address.NOWHERE, 1,
Address.NOWHERE);
            assertNotNull(pr, handler.getClass().getName() + ".process(UnsupportedMessage) return
ed null ");
            assertNotNull(pr.messages,
                    handler.getClass().getName() + ".process(UnsupportedMessage) returned a Proce
ssingResult with a null message list");
            assertFalse(pr.messages.isEmpty(),
                    handler.getClass().getName() + ".process(UnsupportedMessage) returned empty P
rocessingResult");
            assertTrue(pr.messages.get(0) instanceof ErrorResponse,
                    "Content of ProcessingResult was not an ErrorResponse");
            assertTrue(((ErrorResponse) pr.messages.get(0)).getErrorDescription().contains("not s
upported"),
                    "ErrorResponse does not appear to be a 'not supported' error");

        }
    }
}
```

A.541 SSTAF/src/verification/mil.sstaftest.util/src/main/java/mil/sstaftest/util/SimpleMockFeature.java

```
package mil.sstaftest.util;
import mil.sstaf.core.features.BaseFeature;
public class SimpleMockFeature extends BaseFeature {
    /**
     * Constructor for subclasses.
     * <p>
     * Note that concrete implementations must have a no-args constructor to be loadable as servi
ces
     */
    protected SimpleMockFeature() {
        super("Simple Mock Feature",
                1, 0, 0,
                false, "This is nothing");
    }
}
```

A.542 SSTAF/src/verification/mil.sstaftest.util/src/main/java/module-info.java

```
module mil.sstaftest.util {
    exports mil.sstaftest.util;
    requires mil.sstaf.core;

    requires org.junit.jupiter.api;
    opens mil.sstaftest.util to org.junit.platform.commons;
}
```

A.543 SSTAF/src/verification/mil.sstaftest.util/src/main/python/helpers/args.py

```
import sys
s = sys.stdin.readline().strip()
while s not in ['break', 'quit']:
    index = int(s)
    sys.stdout.write(sys.argv[index] + '\n')
    sys.stdout.flush()
    s = sys.stdin.readline().strip()
```

A.544 SSTAF/src/verification/mil.sstaftest.util/src/main/python/helpers/upper.py

```
import sys
s = sys.stdin.readline().strip()
while s not in ['break', 'quit']:
    sys.stdout.write(s.upper() + '\n')
    sys.stdout.flush()
    s = sys.stdin.readline().strip()
```

Appendix B – Source Code for SSTAF Gradle Plugin

B.1 sstafGradlePlugin/build.gradle

```
/**
 * Build configuration for the plugin
 */
plugins {
    // Apply the Java Gradle plugin development plugin to add support for developing Gradle plugi
ns
    id 'java-gradle-plugin'
    id 'maven-publish'
    id 'com.gradle.plugin-publish' version '0.14.0'
}
repositories {
    mavenLocal()
    mavenCentral()
}
dependencies {
    implementation 'org.ow2.asm:asm:9.2'
    implementation 'com.fasterxml.jackson.core:jackson-core:2.13.1'
    implementation 'com.fasterxml.jackson.core:jackson-databind:2.13.1'
    implementation 'com.fasterxml.jackson.core:jackson-annotations:2.13.1'
    // Use JUnit test framework for unit tests
    testImplementation 'org.junit.jupiter:junit-jupiter:5.8.1'
    testRuntimeOnly 'org.junit.jupiter:junit-jupiter:5.8.1'
}
group 'sstaf'
version '1.0'
gradlePlugin {
    // Define the plugin
    plugins {
        sstafPlugin {
            id = 'sstaf'
            displayName = 'SSTAF Module-builder Plugin'
            description = 'Builds SSTAF things'
            implementationClass = 'mil.sstaf.gradle.plugin.SSTAFGradlePlugin'
        }
    }
}
publishing {
    repositories {
        maven {
            name = 'localPluginRepository'
            url = System.getProperty('user.home') + "/.m2/local-plugin-repository"
        }
    }
}
```

B.2 sstafGradlePlugin/settings.gradle

```
/*
 * Specifies the content to build and other root-level settings.
 */
rootProject.name = 'sstaf.gradle.plugin'
```

B.3 sstafGradlePlugin/src/main/java/mil/sstaf/gradle/plugin/SSTAFGradlePlugin.java

```
package mil.sstaf.gradle.plugin;
import mil.sstaf.gradle.plugin.javamodules.ExtraModuleInfoTransform;
import mil.sstaf.gradle.plugin.resourcemgmt.MessageDigestTask;
import org.gradle.api.Plugin;
import org.gradle.api.Project;
import org.gradle.api.artifacts.Configuration;
import org.gradle.api.attributes.Attribute;
import org.gradle.api.plugins.JavaPlugin;
import org.gradle.api.plugins.jvm.internal.JvmEcosystemUtilities;
import javax.inject.Inject;
/**
 * Entry point of our plugin that should be applied in the root project.
 */
abstract public class SSTAFGradlePlugin implements Plugin<Project> {
    JvmEcosystemUtilities utilities;
    @Inject
    public SSTAFGradlePlugin(JvmEcosystemUtilities jvmEcosystemUtilities) {
        this.utilities = jvmEcosystemUtilities;
    }
    @Override
    public void apply(Project project) {
        //
        // Register the plugin extension as 'sstaf {}' configuration block
        //
        SSTAFPluginExtension extension = project.getObjects().newInstance(SSTAFPluginExtension.cl
ass);
        project.getExtensions().add(SSTAFPluginExtension.class, "sstaf", extension);
        //
        // Register tasks
        //
        project.getTasks().register("hashResources", MessageDigestTask.class, task -> {
            task.getHashAlgorithm().set(extension.getHashAlgorithm().getOrElse("MD5"));
        });
        // setup the transform for all projects in the build
        project.getPlugins().withType(JavaPlugin.class).configureEach(javaPlugin -> {
            configureTransform(project, extension);
        });
        //IntegrationTestSetup.apply(project);
    }
    private void configureTransform(Project project, SSTAFPluginExtension extension) {
        Attribute<String> artifactType = Attribute.of("artifactType", String.class);
        Attribute<Boolean> javaModule = Attribute.of("javaModule", Boolean.class);
        // compile and runtime classpath express that they only accept modules by requesting the
javaModule=true attribute
        project.getConfigurations().matching(this::isResolvingJavaPluginConfiguration).all(
                c -> c.getAttributes().attribute(javaModule, true));
        // all Jars have a javaModule=false attribute by default; the transform also recognizes m
odules and returns them without modification
        project.getDependencies().getArtifactTypes().getByName("jar").getAttributes().attribute(j
avaModule, false);
        // register the transform for Jars and "javaModule=false -> javaModule=true"; the plugin
extension object fills the input parameter
        project.getDependencies().registerTransform(ExtraModuleInfoTransform.class, t -> {
            t.parameters(p -> {
                p.setModuleInfo(extension.getModuleInfo());
                p.setAutomaticModules(extension.getAutomaticModules());
            });
            t.getFrom().attribute(artifactType, "jar").attribute(javaModule, false);
            t.getTo().attribute(artifactType, "jar").attribute(javaModule, true);
        });
    }
    private boolean isResolvingJavaPluginConfiguration(Configuration configuration) {
        if (!configuration.isCanBeResolved()) {
```

```
            return false;
        }
        return configuration.getName().endsWith(JavaPlugin.COMPILE_CLASSPATH_CONFIGURATION_NAME.s
ubstring(1))
                || configuration.getName().endsWith(JavaPlugin.RUNTIME_CLASSPATH_CONFIGURATION_NA
ME.substring(1))
                || configuration.getName().endsWith(JavaPlugin.ANNOTATION_PROCESSOR_CONFIGURATION
_NAME.substring(1));
    }
}
```

B.4 sstafGradlePlugin/src/main/java/mil/sstaf/gradle/plugin/SSTAFPluginExtension.java

```
package mil.sstaf.gradle.plugin;
import mil.sstaf.gradle.plugin.javamodules.ModuleInfo;
import org.gradle.api.Action;
import org.gradle.api.provider.Property;
import javax.annotation.Nullable;
import java.util.HashMap;
import java.util.Map;
/**
 * A data class to collect all the module information we want to add.
 * Here the class is used as extension that can be configured in the build script
 * and as input to the ExtraModuleInfoTransform that add the information to Jars.
 */
abstract public class SSTAFPluginExtension {
    private final Map<String, ModuleInfo> moduleInfo = new HashMap<>();
    private final Map<String, String> automaticModules = new HashMap<>();
    abstract public Property<String> getHashAlgorithm();
    /**
     * Add full module information for a given Jar file.
     *
     * @param jarName the name of the jar
     * @param moduleName the module name to assign to the jar
     * @param moduleVersion the version to assign to the module
     */
    public void module(String jarName, String moduleName, String moduleVersion) {
        module(jarName, moduleName, moduleVersion, null);
    }
    /**
     * Add full module information, including exported packages and dependencies, for a given Jar
file.
     *
     * @param jarName the name of the jar
     * @param moduleName the module name to assign to the jar
     * @param moduleVersion the version to assign to the module
     * @param conf a configuration to execute on the generated {@code ModuleInfo}
     */
     /* Add full module information, including exported packages and dependencies, for a given Ja
r file.
     */
    public void module(String jarName, String moduleName, String moduleVersion, @Nullable Action<
? super ModuleInfo> conf) {
        ModuleInfo moduleInfo = new ModuleInfo(moduleName, moduleVersion);
        if (conf != null) {
            conf.execute(moduleInfo);
        }
        this.moduleInfo.put(jarName, moduleInfo);
    }
    /**
     * Add only an automatic module name to a given jar file.
     * @param jarName the name of the jar
     * @param moduleName the module name to assign to the jar
     */
    public void automaticModule(String jarName, String moduleName) {
        automaticModules.put(jarName, moduleName);
    }
    protected Map<String, ModuleInfo> getModuleInfo() {
        return moduleInfo;
    }
    protected Map<String, String> getAutomaticModules() {
        return automaticModules;
    }
}
```

B.5 sstafGradlePlugin/src/main/java/mil/sstaf/gradle/plugin/javamodules/ExtraModuleInfoTransform.java

```
package mil.sstaf.gradle.plugin.javamodules;
import org.gradle.api.artifacts.transform.InputArtifact;
import org.gradle.api.artifacts.transform.TransformAction;
import org.gradle.api.artifacts.transform.TransformOutputs;
import org.gradle.api.artifacts.transform.TransformParameters;
import org.gradle.api.file.FileSystemLocation;
import org.gradle.api.provider.Provider;
import org.gradle.api.tasks.Input;
import org.objectweb.asm.ClassWriter;
import org.objectweb.asm.ModuleVisitor;
import org.objectweb.asm.Opcodes;
import java.io.*;
import java.util.Collections;
import java.util.Map;
import java.util.jar.*;
import java.util.regex.Pattern;
import java.util.zip.ZipEntry;
/**
 * An artifact transform that applies additional information to Jars without module information.
 * The transformation fails the build if a Jar does not contain information and no extra informat
ion
 * was defined for it. This way we make sure that all Jars are turned into modules.
 */
abstract public class ExtraModuleInfoTransform implements TransformAction<ExtraModuleInfoTransfor
m.Parameter> {
    private static void addAutomaticModuleName(File originalJar, File moduleJar, String moduleNam
e) {
        try (JarInputStream inputStream = new JarInputStream(new FileInputStream(originalJar))) {
            Manifest manifest = inputStream.getManifest();
            manifest.getMainAttributes().put(new Attributes.Name("Automatic-Module-Name"), module
Name);
            try (JarOutputStream outputStream = new JarOutputStream(new FileOutputStream(moduleJa
r), inputStream.getManifest())) {
                copyEntries(inputStream, outputStream);
            }
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
    private static void addModuleDescriptor(File originalJar, File moduleJar, ModuleInfo moduleIn
fo) {
        try (JarInputStream inputStream = new JarInputStream(new FileInputStream(originalJar))) {
            try (JarOutputStream outputStream = new JarOutputStream(new FileOutputStream(moduleJa
r), inputStream.getManifest())) {
                copyEntries(inputStream, outputStream);
                outputStream.putNextEntry(new JarEntry("module-info.class"));
                outputStream.write(addModuleInfo(moduleInfo));
                outputStream.closeEntry();
            }
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
    private static void copyEntries(JarInputStream inputStream, JarOutputStream outputStream) thr
ows IOException {
        JarEntry jarEntry = inputStream.getNextJarEntry();
        while (jarEntry != null) {
            outputStream.putNextEntry(jarEntry);
            outputStream.write(inputStream.readAllBytes());
            outputStream.closeEntry();
            jarEntry = inputStream.getNextJarEntry();
        }
    }
```

```
    private static byte[] addModuleInfo(ModuleInfo moduleInfo) {
        ClassWriter classWriter = new ClassWriter(0);
        classWriter.visit(Opcodes.V9, Opcodes.ACC_MODULE, "module-info", null, null, null);
        ModuleVisitor moduleVisitor = classWriter.visitModule(moduleInfo.getModuleName(), Opcodes
.ACC_OPEN, moduleInfo.getModuleVersion());
        for (String packageName : moduleInfo.getExports()) {
            moduleVisitor.visitExport(packageName.replace('.', '/'), 0);
        }
        moduleVisitor.visitRequire("java.base", 0, null);
        for (String requireName : moduleInfo.getRequires()) {
            moduleVisitor.visitRequire(requireName, 0, null);
        }
        for (String requireName : moduleInfo.getRequiresTransitive()) {
            moduleVisitor.visitRequire(requireName, Opcodes.ACC_TRANSITIVE, null);
        }
        moduleVisitor.visitEnd();
        classWriter.visitEnd();
        return classWriter.toByteArray();
    }
    @InputArtifact
    protected abstract Provider<FileSystemLocation> getInputArtifact();
    @Override
    public void transform(TransformOutputs outputs) {
        Map<String, ModuleInfo> moduleInfo = getParameters().moduleInfo;
        Map<String, String> automaticModules = getParameters().automaticModules;
        File originalJar = getInputArtifact().get().getAsFile();
        String originalJarName = originalJar.getName();
        if (isModule(originalJar)) {
            outputs.file(originalJar);
        } else if (moduleInfo.containsKey(originalJarName)) {
            addModuleDescriptor(originalJar, getModuleJar(outputs, originalJar), moduleInfo.get(o
riginalJarName));
        } else if (isAutoModule(originalJar)) {
            outputs.file(originalJar);
        } else if (automaticModules.containsKey(originalJarName)) {
            addAutomaticModuleName(originalJar, getModuleJar(outputs, originalJar), automaticModu
les.get(originalJarName));
        } else {
            throw new RuntimeException("Not a module and no mapping defined: " + originalJarName)
;
        }
    }
    private boolean isModule(File jar) {
        Pattern moduleInfoClassMrjarPath = Pattern.compile("META-INF/versions/\\d+/module-info.cl
ass");
        try (JarInputStream inputStream = new JarInputStream(new FileInputStream(jar))) {
            boolean isMultiReleaseJar = containsMultiReleaseJarEntry(inputStream);
            ZipEntry next = inputStream.getNextEntry();
            while (next != null) {
                if ("module-info.class".equals(next.getName())) {
                    return true;
                }
                if (isMultiReleaseJar && moduleInfoClassMrjarPath.matcher(next.getName()).matches
()) {
                    return true;
                }
                next = inputStream.getNextEntry();
            }
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
        return false;
    }
    private boolean containsMultiReleaseJarEntry(JarInputStream jarStream) {
        Manifest manifest = jarStream.getManifest();
        return manifest != null && Boolean.parseBoolean(manifest.getMainAttributes().getValue("Mu
lti-Release"));
    }
```

```
    private boolean isAutoModule(File jar) {
        try (JarInputStream inputStream = new JarInputStream(new FileInputStream(jar))) {
            return inputStream.getManifest().getMainAttributes().getValue("Automatic-Module-Name"
) != null;
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
    private File getModuleJar(TransformOutputs outputs, File originalJar) {
        return outputs.file(originalJar.getName().substring(0, originalJar.getName().lastIndexOf(
'.')) + "-module.jar");
    }
    public static class Parameter implements TransformParameters, Serializable {
        private Map<String, ModuleInfo> moduleInfo = Collections.emptyMap();
        private Map<String, String> automaticModules = Collections.emptyMap();
        @Input
        public Map<String, ModuleInfo> getModuleInfo() {
            return moduleInfo;
        }
        public void setModuleInfo(Map<String, ModuleInfo> moduleInfo) {
            this.moduleInfo = moduleInfo;
        }
        @Input
        public Map<String, String> getAutomaticModules() {
            return automaticModules;
        }
        public void setAutomaticModules(Map<String, String> automaticModules) {
            this.automaticModules = automaticModules;
        }
    }
}
```

B.6 sstafGradlePlugin/src/main/java/mil/sstaf/gradle/plugin/javamodules/ModuleInfo.java

```
package mil.sstaf.gradle.plugin.javamodules;
import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;
/**
 * Data class to hold the information that should be added as module-info.class to an existing Ja
r file.
 */
public class ModuleInfo implements Serializable {
    private final String moduleName;
    private final String moduleVersion;
    private final List<String> exports = new ArrayList<>();
    private final List<String> requires = new ArrayList<>();
    private final List<String> requiresTransitive = new ArrayList<>();
    public ModuleInfo(String moduleName, String moduleVersion) {
        this.moduleName = moduleName;
        this.moduleVersion = moduleVersion;
    }
    public void exports(String exports) {
        this.exports.add(exports);
    }
    public void requires(String requires) {
        this.requires.add(requires);
    }
    public void requiresTransitive(String requiresTransitive) {
        this.requiresTransitive.add(requiresTransitive);
    }
    public String getModuleName() {
        return moduleName;
    }
    protected String getModuleVersion() {
        return moduleVersion;
    }
    protected List<String> getExports() {
        return exports;
    }
    protected List<String> getRequires() {
        return requires;
    }
    protected List<String> getRequiresTransitive() {
        return requiresTransitive;
    }
}
```

B.7 sstafGradlePlugin/src/main/java/mil/sstaf/gradle/plugin/resourcemgmt/MessageDigestTask.java

```
package mil.sstaf.gradle.plugin.resourcemgmt;
import org.gradle.api.DefaultTask;
import org.gradle.api.file.FileCollection;
import org.gradle.api.provider.Property;
import org.gradle.api.tasks.*;
import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.security.NoSuchAlgorithmException;
/**
 * Task that generates {@code MessageDigest} reports for resources
 */
public abstract class MessageDigestTask extends DefaultTask {
    private final FileCollection sourceFiles;
    /**
     * Constructor
     */
    public MessageDigestTask() {
        super();
        setGroup("other");
        dependsOn("processResources");
        File bd = getProject().getBuildDir();
        String resourcesPath = bd.getAbsolutePath() + File.separator + "resources" + File.separat
or + "main";
        File resourcesDir = new File(resourcesPath);
        sourceFiles = this.getProject().fileTree(resourcesPath);
        doFirst(task -> {
            if (resourcesDir.exists()) {
                String ha = getHashAlgorithm().getOrElse("MD5");
                try {
                    Path basePath = Path.of(resourcesPath);
                    Files.walkFileTree(basePath, new MessageDigestVisitor(basePath, ha));
                } catch (NoSuchAlgorithmException | IOException e) {
                    e.printStackTrace();
                }
            }
        });
    }
    @Input
    abstract public Property<String> getHashAlgorithm();
    @SkipWhenEmpty
    @InputFiles
    @PathSensitive(PathSensitivity.NONE)
    public FileCollection getSourceFiles() {
        return this.sourceFiles;
    }
}
```

B.8 sstafGradlePlugin/src/main/java/mil/sstaf/gradle/plugin/resourcemgmt/MessageDigestVisitor.java

```
package mil.sstaf.gradle.plugin.resourcemgmt;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.ObjectWriter;
import com.fasterxml.jackson.databind.node.ObjectNode;
import java.io.*;
import java.nio.file.FileVisitResult;
import java.nio.file.FileVisitor;
import java.nio.file.Path;
import java.nio.file.attribute.BasicFileAttributes;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
/**
 * Implementation of {@code FileVisitor} that generates {@code MessageDigest} checksums for each
file
 * in a directory tree.
 *
 * <p>
 * This class generates a checksum report for each directory in the directory graph. The report i
s encoded
 * in JSON and is written to a file that is always named "checksums.json."
 * </p>
 *
 * <p>
 * The hashing algorithm is specified in the constructor. If the algorithm is null or zero-length
, the
 * class falls back to using MD5. The algorithm used to generate the checksums is recorded in the
"checksums.json"
 * file. The property name for the algorithm is "__hashAlgorithm".
 * </p>
 */
class MessageDigestVisitor implements FileVisitor<Path> {
    private final Path resourceDir;
    private final String hashAlgorithm;
    private final MessageDigest messageDigest;
    private final ObjectNode hashReport;
    private final ObjectMapper objectMapper;
    private int depth;
    /**
     * Constructor
     *
     * @param resourceDir   the top-level directory for the tree walk. This is used to compute re
lative paths
     *                      for sub-directories.
     * @param hashAlgorithm the hashing algorithm to use.
     * @throws NoSuchAlgorithmException if a non-existent algorithm is selected.
     */
    public MessageDigestVisitor(final Path resourceDir, final String hashAlgorithm) throws NoSuch
AlgorithmException {
        this.hashAlgorithm = hashAlgorithm == null ? "MD5" : hashAlgorithm;
        this.resourceDir = resourceDir;
        messageDigest = MessageDigest.getInstance(this.hashAlgorithm);
        Path reportFilePath = Path.of(resourceDir.toString(), "checksums.json");
        File oldReportFile = reportFilePath.toFile();
        if (oldReportFile.exists()) {
            final boolean delete = oldReportFile.delete();
            if (!delete) {
                System.err.println("sss");
            }
        }
        objectMapper = new ObjectMapper();
        hashReport = objectMapper.createObjectNode();
        hashReport.put("__hashAlgorithm", hashAlgorithm);
        depth = 0;
```

```
    }
    /**
     * Method executed at the beginning of directory traversal.
     *
     * <p>
     * Upon entering a new directory, a new {@code JSONObject} is created to hold the checksum re
port.
     * The "__hashAlgorithm" and "__path" properties are added to it and the {@code JSONObject) i
s
     * pushed onto a stack. The stack maintains separate reports for each directory in the tree.
     * </p>
     *
     * @param dir   the {@code Path} to the directory being visited
     * @param attrs file attributes (not used)
     * @return {@code FileVisitResult.CONTINUE}
     */
    @Override
    public FileVisitResult preVisitDirectory(Path dir, BasicFileAttributes attrs) {
        ++depth;
        return FileVisitResult.CONTINUE;
    }
    /**
     * Method executed for each file.
     * <p>
     * Given a {@code Path} to a file, this method reads the contents of the file and routes them
     * into a {@code MessageDigest}. The {@code MessageDigest} then generates a digest (checksum)
of
     * the bytes from the file. This value is then serialized into a String and recorded in the
     * hash report JSONObject.
     * </p>
     *
     * @param file  the {@code Path} to the file being visited
     * @param attrs file attributes (not used)
     * @return {@code FileVisitResult.CONTINUE}
     * @throws IOException if the file can not be read.
     */
    @Override
    public FileVisitResult visitFile(Path file, BasicFileAttributes attrs) throws IOException {
        Path relative = resourceDir.relativize(file);
        byte[] buffer = new byte[1024];
        int read = 0;
        messageDigest.reset();
        InputStream inputStream = new FileInputStream(file.toFile());
        while ((read = inputStream.read(buffer)) > 0) {
            messageDigest.update(buffer, 0, read);
        }
        byte[] checksum = messageDigest.digest();
        String encoded = bytesToString(checksum);
        hashReport.put(relative.toString(), encoded);
        return FileVisitResult.CONTINUE;
    }
    /**
     * Method executed when something goes wrong.
     *
     * <p>Terminates traversal of the directory tree</p>
     *
     * @param file the {@code Path} to the file or directory that caused the problem.
     * @param exc  the {@code IOException=} that was thrown
     * @return {@code FileVisitResult.TERMINATE}
     */
    @Override
    public FileVisitResult visitFileFailed(Path file, IOException exc) {
        System.err.println("Failed to process: " + file.toAbsolutePath());
        exc.printStackTrace(System.err);
        return FileVisitResult.TERMINATE;
    }
    /**
```

```
     * Method executed when a directory has been processed completely. This includes all subdirec
tories.
     *
     * <p>
     * When directory traversal is complete, the hash report for the current directory is popped
from
     * the directory stack and written to the "checksums.json" file.
     * </p>
     *
     * @param dir the {@code Path} to the directory being visited
     * @param exc Any {@code IOException} thrown during traversal.
     * @return {@code FileVisitResult.CONTINUE}
     * @throws IOException if the report can not be written.
     */
    @Override
    public FileVisitResult postVisitDirectory(Path dir, IOException exc) throws IOException {
        if (exc == null) {
            --depth;
            if (depth == 0) {
                Path reportFilePath = Path.of(dir.toString(), "checksums.json");
                FileWriter fileWriter = new FileWriter(reportFilePath.toFile());
                fileWriter.write(hashReport.toPrettyString());
                fileWriter.flush();
                fileWriter.close();
            }
            return FileVisitResult.CONTINUE;
        } else {
            System.err.println("Failed to process: " + dir.toAbsolutePath());
            exc.printStackTrace(System.err);
            return FileVisitResult.TERMINATE;
        }
    }
    /**
     * Converts the byte array from the {@code MessageDigest} into a hex string.
     *
     * @param bytes the bytes
     * @return a {@code String} containing hexadecimal values.
     */
    private String bytesToString(byte[] bytes) {
        StringBuilder hexString = new StringBuilder();
        for (byte aByte : bytes) {
            if ((0xff & aByte) < 0x10) {
                hexString.append("0").append(Integer.toHexString((0xFF & aByte)));
            } else {
                hexString.append(Integer.toHexString(0xFF & aByte));
            }
        }
        return hexString.toString();
    }
}
```

Appendix C – Source Code for File Extraction Tool

```
public class Extractor {
    public static void main(String[] args) {
        if (args.length < 3) {
            printUsage();
            System.exit(2);
        }

        String file = args[0];
        String outputDir = args[1];
        String prefix = args[2];

        try (BufferedReader inputReader = new BufferedReader(new InputStreamReader(new FileInputS
tream(file)))) {
            BufferedWriter outputWriter = null;
            String line;
            while ((line = inputReader.readLine()) != null) {
                if (line.contains(prefix)) {
                    if (outputWriter != null) {
                        outputWriter.flush();
                        outputWriter.close();
                    }
                    // Eliminate the subsection labels.
                    String[] fields = line.split("\\s+");
                    String pathString = fields[1];
                    Path outputPath = Path.of(outputDir, pathString);
                    Files.createDirectories(outputPath.getParent());
                    outputWriter = new BufferedWriter(new OutputStreamWriter(new FileOutputStream
(outputPath.toFile())));
                    System.out.println("Writing to " + outputPath);
                } else {
                    if (outputWriter != null) {
                        outputWriter.write(line);
                        outputWriter.newLine();
                        outputWriter.flush();
                    }
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
            System.exit(1);
        }
    }


private static void printUsage() {
        System.err.println("Usage: java org.example.Extractor inputFile outputDir pathPrefix");
}
```

BIBLIOGRAPHY

Bloch, J., Bowbeer, J., Lea, D., Holmes, D., Peierls, T., & Goetz, B. (2006). *Java concurrency in practice*. Pearson Education.

Helm, R., Vlissides, J., Gamma, E., & Johnson, R. (1994). *Design patterns: elements of reusable object-oriented software*. Pearson Education.

Martin, R. C. (2009). *Clean code: a handbook of agile software craftsmanship*. Prentice Hall.

Pressman, R. S. (2001). *Software engineering: a practitioner's approach* (5th ed.). McGraw Hill.

Schmidt, D., Stal, M., Rohnert, H., & Buschman, F. (2000). *Pattern-oriented software architectures: patterns for concurrent and networked objects*. John Wiley & Sons.

LIST OF ACRONYMS

ANSUR	Anthropometric Survey of U.S. Army personnel
CSV	comma-separated value
DAC	DEVCOM Analysis Center
DEVCOM	U.S. Army Combat Capabilities Development Command
DOD	Department of Defense
JSON	JavaScript Object Notation
JVM	Java virtual machine
no-arg	no-argument
OneSAF	One Semi-Automated Force
ORCA	Operational Requirements-based Casualty Assessment
REPL	Read-Evaluate-Print Loop
SSTAF	Soldier and Squad Trade Space Analysis Framework
stdin	standard input
stdout	standard output

ORGANIZATION

DEVCOM Analysis Center
FCDD-DAW-W/ R. Bowers
FCDD-DAW-W/J. Ehlenberger
FCDD-DAW-W/T. Myers
FCDD-DAW-G/J. Collins
FCDD-DAW-G/J. Gantert
FCDD-DAW-G/Z. Steelman
FCDD-DAG-S/J. Way
6896 Mauchly St.
Aberdeen Proving Ground, MD 21005-5071

DEVCOM Army Research Laboratory
FCDD-RLD-DCI/Tech Library
2800 Powder Mill Rd.
Adelphi, MD 20783

Defense Technical Information Center
ATTN: DTIC-O
8725 John J. Kingman Rd.
Fort Belvoir, VA 22060-6218