**TRAC-M-TR-16-05**
**06 October 2015**

# COMBATXXI, JDAFS, and LBC Integration Requirements for EASE

**TRADOC Analysis Center**
**700 Dyer Road**
**Monterey, California 93943-0692**

**This study cost the Department of Defense approximately $139,000 expended by TRAC in Fiscal Years 14-15.**
**Prepared on 20151006**
**TRAC Project Code # 060118.**

**TRAC-M-TR-16-05**
**06 October 2015**

# COMBATXXI, JDAFS, and LBC Integration Requirements for EASE

**Cardy Moten III**
**Arnie Buss**
**Sam Buttrey**

**TRADOC Analysis Center**
**700 Dyer Road**
**Monterey, California 93943-0692**

**This page intentionally left blank.**

# REPORT DOCUMENTATION PAGE

*Form Approved*
*OMB No. 0704-0188*



| 1. REPORT DATE *(DD-MM-YYYY)* | 2. REPORT TYPE | 3. DATES COVERED *(From - To)* |
|---|---|---|
| | Technical Report | SEP 2014 - SEP 2015 |

**4. TITLE AND SUBTITLE**

COMBATXXI, JDAFS, and LBC Integration Requirements for the Executable Architecture Systems Engineering (EASE)

**5a. CONTRACT NUMBER**

**5b. GRANT NUMBER**

**5c. PROGRAM ELEMENT NUMBER**

**6. AUTHOR(S)**

MAJ Cardy Moten III

**5d. PROJECT NUMBER**

060118

**5e. TASK NUMBER**

**5f. WORK UNIT NUMBER**

**7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**

TRADOC Analysis Center, Monterey, CA
Naval Postgraduate School, Monterey, CA

**8. PERFORMING ORGANIZATION REPORT NUMBER**

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

USACE Engineer Research Development Center, Vicksburg, MS

**10. SPONSOR/MONITOR'S ACRONYM(S)**

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

TRAC-M-TR-16-05

**12. DISTRIBUTION / AVAILABILITY STATEMENT**

Approve for public release; distribution is unlimited. This determination was made on 04 October 2015.

**13. SUPPLEMENTARY NOTES**

**14. ABSTRACT**

The purpose of this project is to assist the U.S. Army Corps of Engineers Research and Development Center (ERDC) and the Army Research Laboratory (ARL) in the identification of low and high level requirements to integrate Combine Arms Analysis Tool for the 21st Century (COMBATXXI), Logistics Battle Command (LBC), and Joint Dynamic Allocation of Fires and Sensors (JDAFS) with a cloud-based simulation manager called the Executable Architecture for Systems Engineering (EASE). Using EASE will give an analyst the capability to use the appropriate modeling tool during a tradespace analysis. In order to integrate a combat model simulation tool with EASE, the EASE system engineers have to know the model's operating requirements and customizable parameters. Operating requirements are defined as the elements that are critical to the proper execution of a combat model. Customizable parameters are the options within a scenario file the user can modify for a specific purpose. In order to determine the EASE integration requirements for each model, the study team had to develop a use case scenario, operate each model with an example scenario, and develop appropriate batch scripts to execute the model automatically. The results of our analysis show that it is possible to integrate COMBATXXI, LBC, and JDAFS with EASE. Because COMBATXXI is a large and complex model, there will likely be some support requirements from TRAC-WSMR during the initial connection phase with EASE. The most likely users of COMBATXXI within EASE will be the current users of the model, therefore, we do not anticipate an unusual surge in technical support from TRAC-WSMR outside of what is already implemented in their current workload capacity. LBC and JDAFS are more self-contained models and will require less technical support during installation and execution within EASE.

**15. SUBJECT TERMS**

COMBATXXI, LBC, EASE, platform as a service (PAAS), cloud computing, combat model

| 16. SECURITY CLASSIFICATION OF: | | | 17. LIMITATION OF ABSTRACT | 18. NUMBER OF PAGES | 19a. NAME OF RESPONSIBLE PERSON |
|---|---|---|---|---|---|
| a. REPORT | b. ABSTRACT | c. THIS PAGE | | | Cardy Moten III |
| U | U | U | SAR | | 19b. TELEPHONE NUMBER *(include area code)* (831) 656-2452 |

**Standard Form 298 (Rev. 8-98)**
**Prescribed by ANSI Std. Z39.18**

**This page intentionally left blank.**

# Acknowledgments

The authors would like to acknowledge the COMBATXXI technical support team at TRAC-WSMR. A special thanks goes to Kyle Quinnel whose support helped us to understand how to use COMBATXXI from the command line and to connect to the COMBATXXI database. We would also like to thank Dr. Simon Goerger for helping us to understand the Engineered Resilience process.

**This page intentionally left blank.**

# Table of Contents

**This page intentionally left blank.**

# List of Figures

## List of Figures



[1]Jefferey P. Holland. *Engineered Resilent Systems (ERS)*. Presentation to Ms. Blechinger. 2014.

[2]Jefferey P. Holland. *Engineered Resilent Systems (ERS)*. Presentation to Ms. Blechinger. 2014.

[3]TRADOC Analysis Center-White Sands Missile Range. *COMBATXXI Users Guide*. 2014. URL: `https://cxxi.wsmr.army.mil/Welcome.html`.

[4]Peter A Nesbitt et al. *Knowledge Representation for Decision Making Agents*. Tech. rep. ADA589932, XA, TRAC-WSMR*, TRAC-M-TR-13-054, ARMY TRADOC ANALYSIS CENTER MONTEREY CA, July 2013, p. 88.

**This page intentionally left blank.**

# List of Tables

## List of Tables

# Executive Summary

## Background

The Department of Defense (DoD) established Engineered Resilient Systems (ERS) as a Community of Interest in 2011. The purpose of ERS is to integrate technologies that will change the current acquisitions process from process-driven to data-driven with a goal of improving requirements generation, analysis of alternatives, and lifecycle cost requirements. In order to achieve this goal, DoD has chartered the U.S. Army Corps of Engineers Research and Development Center (ERDC) to develop the suite of tools necessary to implement ERS with a focus on tradespace analysis. Since the use of modeling and simulation tools are a critical aspect of tradespace analysis, the requirement for a tool to integrate the operation of multiple simulation models in a Platform as a Service (PaaS) cloud computing environment across the DoD became necessary.

The proposed tool for this operation is called the Executable Architecture for Systems Engineering (EASE). EASE is a cloud-based simulation management tools that seeks to give analysts the ability to use multiple simulation models in the conduct of their studies. The focus of this project is to develop the requirements of integrating EASE with the TRAC-owned models Combined Arms Analysis Tool for the 21st Century (COMBATXXI), Logistics Battle Command (LBC), and Joint Dynamic Allocation of Fires and Sensors (JDAFS). In order to integrate a combat model simulation tool with EASE, the EASE system engineers have to know the model's operating requirements and customizable parameters. Operating requirements are defined as the elements that are critical to the proper execution of a combat model. Customizable parameters are the options within a scenario file the user can modify for a specific purpose. In order to determine the EASE integration requirements for each model, the study team had to develop a use case scenario, operate each model with an example scenario, and develop appropriate batch scripts to execute the model automatically.

## Results

The results of our analysis show that it is possible to integrate COMBATXXI, LBC, and JDAFS with EASE. Because COMBATXXI is a large and complex model, there will likely be some support requirements from TRAC-WSMR during the initial connection phase with EASE. The most likely users of COMBATXXI within EASE will be the current users of the model, therefore, we do not anticipate an unusual surge in technical support from TRAC-WSMR outside of what is already implemented in their current workload capacity. LBC

and JDAFS are more self-contained models and will require less technical support during installation and execution within EASE.

# COMBATXXI, JDAFS, and LBC Integration Requirements for EASE
## Chapter 1
## Introduction

The purpose of this project is to assist the U.S. Army Corps of Engineers' Engineer Research Development Center (ERDC) and the Army Research Laboratory (ARL) in the identification of low and high level requirements to integrate the Combined Arms Analysis Tool for the 21st Century (COMBATXXI), JDAFS, and LBC into a PaaS cloud computing simulation manager called the Executable Architecture Systems Engineering (EASE). This document describes work performed in both FY 2014 and 2015.

## Background

### Engineered Resilient Systems (ERS)

The Secretary of Defense, in 2011, established Engineered Resilient Systems (ERS) as a Community of Interest (COI) chartered with developing "engineering concepts, science, and design tools to protect against malicious compromise of weapon systems and to develop agile manufacturing for trusted and assured defense systems".[1] ERDC is in charge of the ERS community, and a goal of ERS is to develop a prototype toolkit of systems that automate the analysis of the alternatives trade space in the in the acquisitions process resulting in weapon and defense systems that are effective in various configurations and operating environments.[2]

In order to execute this process, the current Joint Capabilities Integration Development System (JCIDS) process, shown in Figure 1, must change to a data-driven process shown in Figure 2. Figure 1 represents the current JCIDS process as linear and as new data is made available, any previous analysis is obsolete and has to start the process over again. Figure 2 proposes a method to solve this dilemma by having all pertinent participants enter their data into a cloud based environment that allows for a deeper and rich analysis of the trade space to include system requirements, alternatives, costs, and operational scenarios.[3]

[1] Secretary of Defense Robert M. Gates. *Science and Technology (S&T) Priorities for Fiscal Years 2013-17 Planning. Memorandum for Secretaries of the Military Departments.* Ed. by Washington D.C. Washington D.C., 2011. URL: `http://www.acq.osd.mil/chieftechnologist/publications/docs/OSD%2002073-11.pdf`.

[2] Eric Spero et al. "A Research Agenda for Tradespace Exploration and Analysis of Engineered Resilient Systems". In: *Procedia Computer Science* 28 (2014), pp. 763–772.

[3] Ibid.

This proposed change to the current JCIDS process resulted from a thorough analysis of the future operating environment and the realization that the Department of Defense (DoD) acquisition process must be streamlined to meet the operational demands of the future operating environment.[4]

Additionally, this new data-driven process will provide the capability for more collaboration between all pertinent stakeholders in the acquisition process by providing relevant and timely data when it is needed, which allows for the analysis of system alternatives in more detail throughout all phases of the acquisition process.[5] In order to utilize this new data-driven process, various analysis, simulation, and other data management tools must be integrated into a cloud-based environment. This report will focus on the requirements to integrate the simulation tools of COMBATXXI, JDAFS, and LBC into the simulation management system EASE.

## Executable Architecture Systems Engineering (EASE)

Recently, the Georgia Tech Research Institute (GTRI) and ARL conducted a demonstration of how to integrate simulation and trade space analysis tools using theFramework for Assessing Cost and Technology (FACT) and EASE software platforms.[6] FACT is used to develop and analyze measures of performance, while EASE provides a user interface to multiple simulations that assess measures of effectiveness with the given measures of performance. The results from EASE are then sent back to FACT for follow on analysis. A key capability of EASE that requires further development is integrating the modeling and simulation capability of distributed simulation models like COMBATXXI, JDAFS, and LBC. This will provide an analyst the capability to choose the right tool at the appropriate step in the trade space exploration process.[7]

## COMBATXXI

COMBATXXI is an entity based modeling and simulation application used by Army TRADOC to design, develop and evaluate CONOPs and TTPs. COMBATXXI is a robust simulation tool that lends itself to reusable entities, entity behaviors and vignettes for brigade level and below combat for Army and Marine forces. What makes COMBATXXI unique and complex is that all aspects of the model input and output have to be built and managed by the user. Figure 3 shows a simplified flow of this process. First, the user will develop all agents,

[4]Idem, "A Research Agenda for Tradespace Exploration and Analysis of Engineered Resilient Systems", op. cit.; Simon R Goerger, Azad M Madni, and Owen J Eslinger. "Engineered resilient systems: a dod perspective". In: *Procedia Computer Science* 28 (2014), pp. 865–872; Eric Spero et al. "Tradespace exploration for the engineering of resilient systems". In: *Procedia Computer Science* 28 (2014), pp. 591–600.

[5]Idem, "Tradespace exploration for the engineering of resilient systems", op. cit.

[6]Tommer Ender et al. *Enterprise Architecture Tradespace Analysis*. Tech. rep. ADA603143, XD, DOD, SERC-2014-TR-043, H98230-08-D-0171. Georgia Instittute of Technology, Atlanta, GA, Feb. 2014, p. 21.

[7]Spero et al., "A Research Agenda for Tradespace Exploration and Analysis of Engineered Resilient Systems", op. cit.; Spero et al., "Tradespace exploration for the engineering of resilient systems", op. cit.

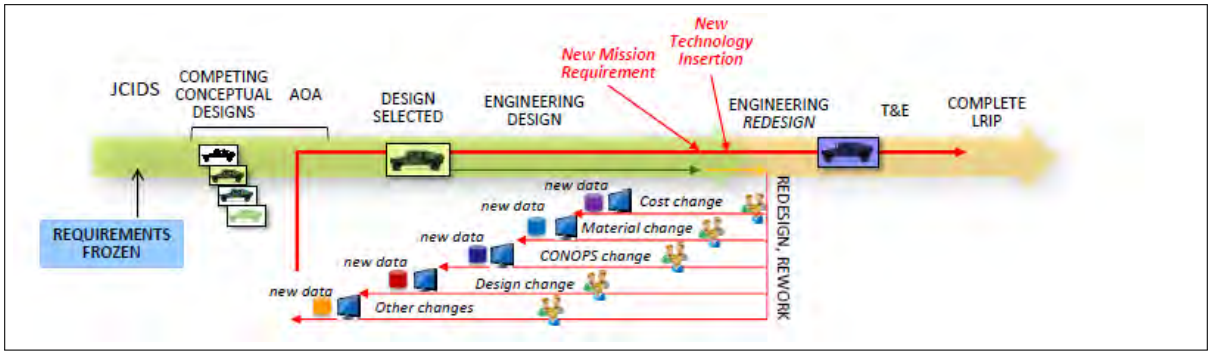



**Figure 1. Current JCIDS Process Flow Diagram from.[8]**

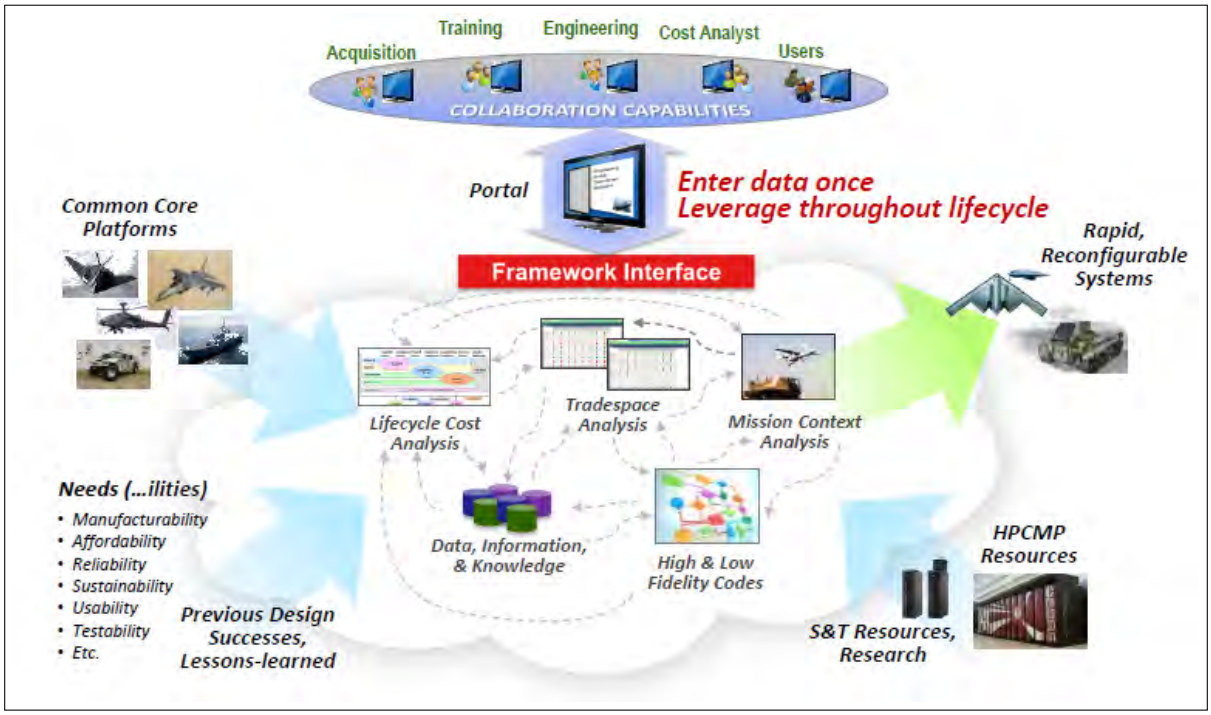



**Figure 2. ERS Process Flow Diagram from.[9]**

behaviors, terrain, networks, orders, and meta data for the simulation using the Scenario Integration Tool Suite (SITS). Once the scenario is built in SITS, multiple databases and Extensible Markup Language (XML) files are created for the core COMBATXXI model to utilize in the simulation. The next step in the process flow, is the for model to execute. The model will continually reference the appropriate database and XML files to adjudicate actions within the simulation environment, and specific events the user selects are stored in logger files. The logger files are text files that store the events that happened within the simulation. COMBATXXI has over 70 types of loggers and each each potentially 50MB in size.[10] Finally, users will collect all of the logger files into a database and proceed to analyze this data in the analysis software of their choice.

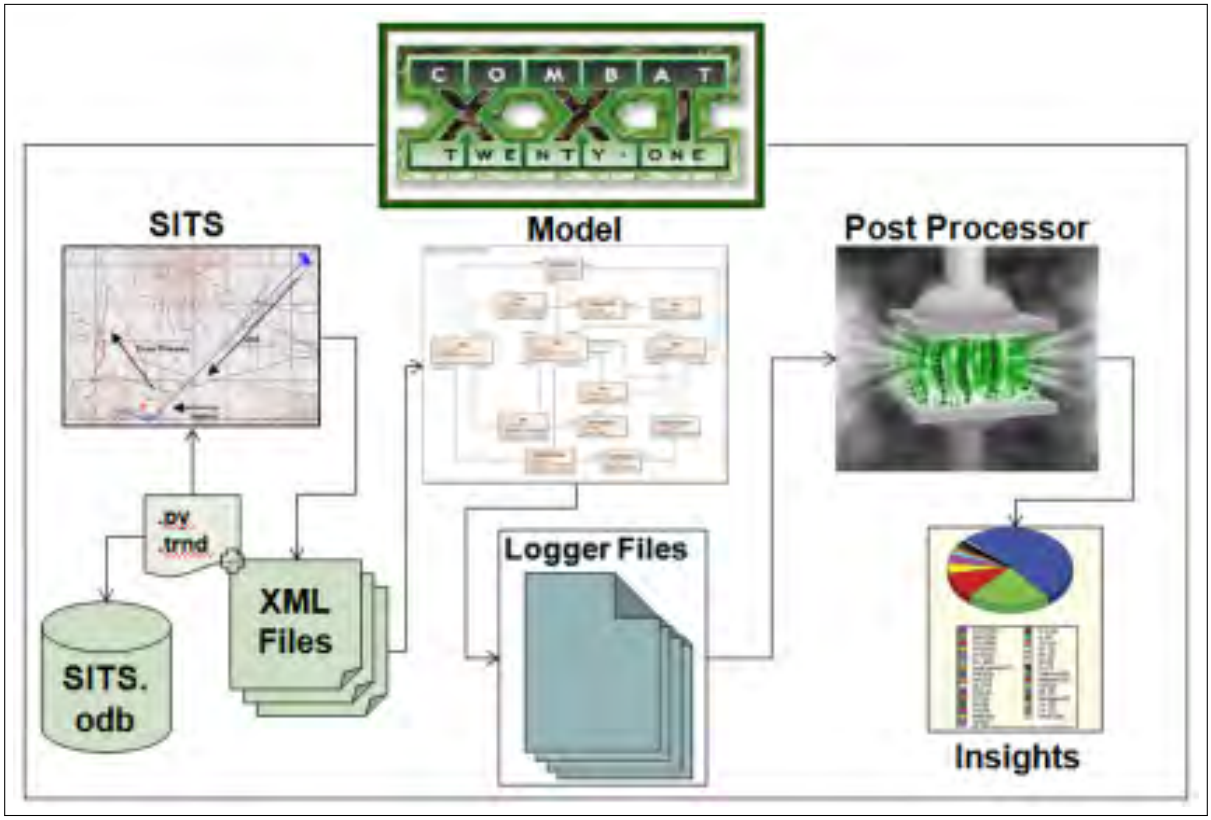



**Figure 3. COMBATXXI Process Flow Diagram from.**[11]

## JDAFS

JDAFS is an entity-level combat simulation model developed at Naval Postgraduate School and TRAC-Monterey. It is a low-resolution model in the sense that entities do not have the same level of detail as in higher resolution combat models (such as COMBATXXI). Its main advantages are speed of scenario development, speed of execution times, relatively modest input requirements, and flexibility. Figure 4 displays an overview of the model.

[10]TRADOC Analysis Center-White Sands Missile Range. *COMBATXXI Users Guide.* 2014. URL: `https://cxxi.wsmr.army.mil/Welcome.html`.

Because of the relative simplicity of JDAFS in both its modeling world-view and its relatively light input requirements, JDAFS is a particularly appropriate model to integrate into EASE. JDAFS scenarios can be built very rapidly and many replications can be executed quickly in order to produce good statistical results.

Of course, JDAFS relatively low-level of resolution (compared with models such as COMBATXXI) means that its results are typically considered to be exploratory in nature. Nevertheless, there are considerable benefits in performing such exploration prior to conducting a large-scale simulation study using a high-resolution model. In most cases the preliminary JDAFS analysis is not inconsistent with the ultimate high-resolution experiments.

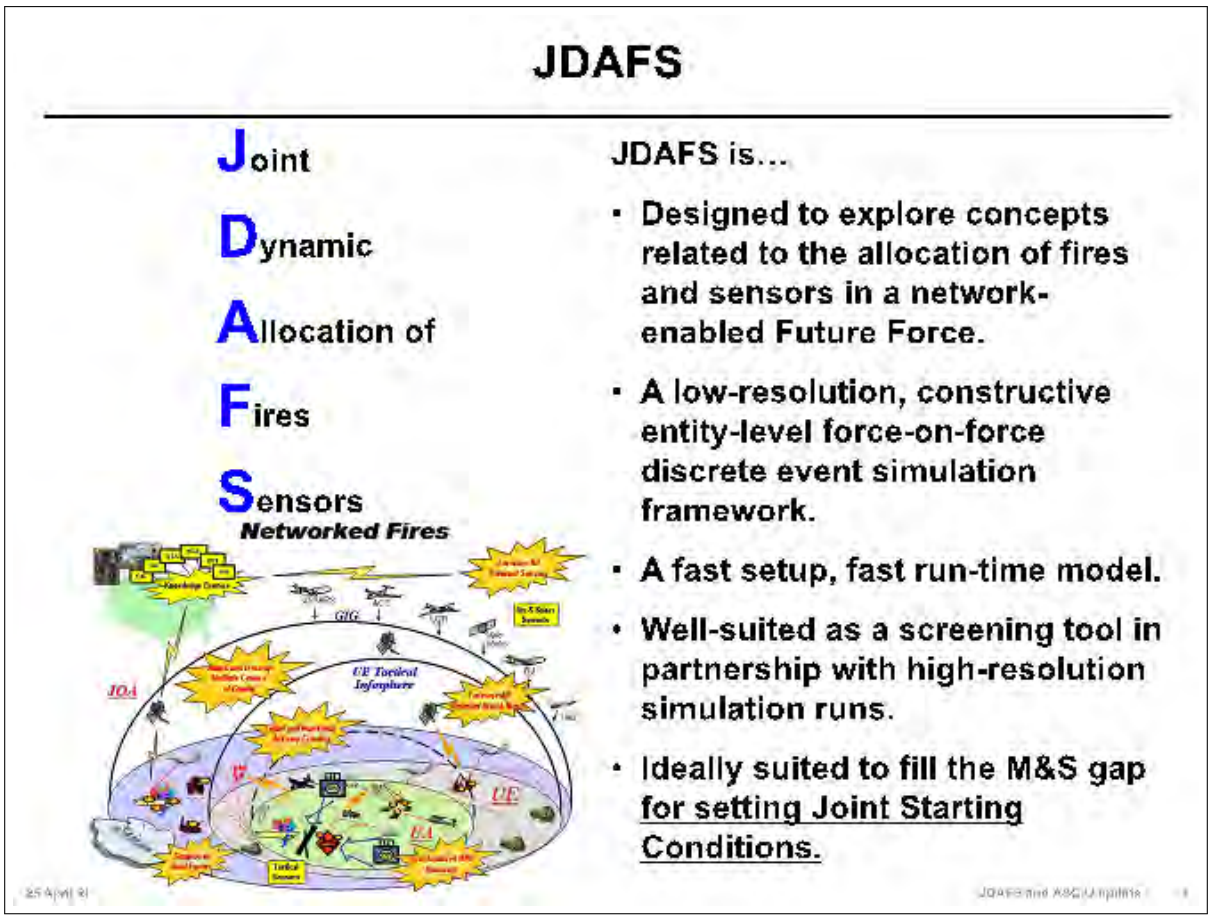



**Figure 4. Overview of JDAFS.**

## LBC

LBC is a closed-form, stochastic simulation model. LBC was developed at TRAC-Monterey and is currently under further development. LBC is used to dynamically forecast and model logistical demand for several classes of supplies. LBC is also capable of modeling networks of supplies and is useful in helping define the logistical Operations Tempo (OPTEMPO) and

Operational Mode (OPMODE) data to be collected.[12] LBC can be run in either stand-alone mode or connected to a high-resolution combat model, such as COMBATXXI. For this initial integration with EASE, only the stand-alone mode will be considered. Because of its ability to be easily executed from the command prompt, the simplicity of the single-file data input, and the ability to write output files to a single location and compress them into a single file for the user's subsequent analysis, LBC is an excellent candidate for inclusion into the EASE framework.

# Key Terminology

We list below key terms used throughout this project.

## Actor

**Actor** is defined as the user or developer of the combat model.[13]

## Conditions

**Conditions** are defined as the circumstances related to pre and post execution of the scenario.[14]

## Customizable parameters

**Customizable parameters** are the options within a scenario file the user can modify for a specific purpose.

## Extensions

**Extensions** are defined as potential contingencies for the scenario integrator or EASE system engineer to manage.[15]

## Main success scenario

**Main success scenario** is defined as what would happen if the scenario executed with no errors.[16]

[12]TRADOC Analysis Center-Fort Lee. *DM/LBC Users Guide v.3.15.0.* 2012.
[13]Cockburn Alistair. *Writing effective use cases.* Boston, MA: Addison-Wesley, 2001, p. 2.
[14]Ibid., p. 2.
[15]Ibid., p. 3.
[16]Ibid., p. 3.

### Operating requirements

**Operating requirements** are defined as the elements that are critical to the proper execution of a combat model.

### Scope

**Scope** is defined as the system(s) related to the use case.[17]

### Stakeholder

**Stakeholder** is defined as someone who has a vested interest in the proper execution of the combat model in EASE.[18]

### Use Case

**Use Case** is defined as the document describing how a user will interact with EASE and a combat model to accomplish a specified goal.[19]

## Problem Statement

The focus of this research was to identify the requirements to integrate COMBATXXI, JDAFS, and LBC into a PaaS cloud computing environment.

### Issues for Analysis

In order to accomplish this purpose, this research addressed the following questions:

**Issue 1:** What are the operating requirements for the simulation models?
- EEA 1.1: What are the operating requirements for COMBATXXI?
- EEA 1.2: What are the operating requirements for JDAFS?
- EEA 1.3: What are the operating requirements for LBC?

**Issue 2:** What are the customizable parameters for the simulation models?
- EEA 2.1: What are the customizable parameters for COMBATXXI?
- EEA 2.2: What are the customizable parameters for JDAFS?
- EEA 2.3: What are the customizable parameters for LBC?

[17]Ibid., p. 2.
[18]Ibid., p. 2.
[19]Ibid., p. 2.

## Literature Review

Gaughan et al., developed EASE as a PaaS tool for analysts to execute combat simulation models without having to concern themselves with the process of building the model themselves.[20] Although there are significant benefits to having multiple combat models across DoD available to a user, Murphy et al., also documented the challenges system engineers face in trying to operate these various models together.[21] Despite these challenges, the EASE developers successfully integrated a few combat models. Gaughan et al., used the One Semi-Automated Forces (OneSAF) simulation model with a route clearance scenario in which the user could enter the probability of detection for a UAV and analyze its output.[22] This demonstrated the ability of EASE to work with the trade space analysis tool FACT. Gaughan et al., worked with the Maneuver Support Battle Laboratory (MSBL) in Fort Leonard Wood, MO to implement their base protection simulation software into EASE.[23] The results of this work afforded the MSBL staff to run their simulation software without having to worry about having enough hardware within their computer lab. The work in this project will focus on the requirements that EASE system engineers will have to meet in order to integrate complex simulation models like COMBATXXI, JDAFS, and LBC.

[20] Chris Gaughan et al. "Executable Architecture Systems Engineering". In: *80th MORS Symposium, Working Group 29*. 2013.

[21] Shaun Murphy et al. "U.S. Army Modeling and Simulation Executable Architecture Deployment Cloud Virtualization Strategy". In: *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*. IEEE, 2012, pp. 880–885; Shaun Murphy, Manuel Diego, and Scott Gallant. "U.S. Army Modeling and Simulation Executable Architecture Deployment Cloud Using Virtualization Technologies to Provide and Extensible Platform as a Service (PaaS) Cloud for Federated Application Configuration and Execution". In: *Fall Simulation Interoperability Workshop, 11F-SIW-051*. 2011.

[22] Chris Gaughan et al. "When Tradespace Analysis Met Combat Modeling and Simulation". In: *I/ITSEC, Session Number: 14264*. 2014.

[23] Chris Gaughan et al. "Bringing Next Generation Simulation into the Land of Practicality". In: *The Interservice/Industry Training, Simulation & Education Conference (I/ITSEC)*. vol. 2013. NTSA, 2013. Chap. 1.

# Chapter 2
# Methodology

The focus of this chapter is the methodology used to determine the EASE integration requirements for COMBATXXI, JDAFS, and LBC.

## COMBATXXI

Appendix B shows the use case that describes the conditions we assume for the use of COMBATXXI within EASE. To determine the operating requirements and customizable parameters for COMBATXXI, we took the following steps:

- Install the latest stable build.
- Build the COMBATXXI scenario file.
- Develop a batch script to automate the model runs.
- Develop a postprocessing method.

### Installation

Since COMBATXXI has a limited distribution, only authorized DoD agencies and contractors can receive a copy of the latest stable build with an official request to the Director, Training and Doctrine Command Analysis Center (TRAC). The requesting agency should also request access to the COMBATXXI wiki site in order to read the latest updates to the model, find example scenarios, and submit trouble tickets.[24] The user can install COMBATXXI in either a Windows or Linux environment; additionally, the stable build also comes with the source code for the model to allow developers to modify COMBATXXI to meet their needs. The user should also have at least 2 GB of memory and will require a 64-bit JRE. Since most users will utilize SITS, in COMBATXXI, the user should install the Java Development Kit (JDK) instead.[25] During initial setup, a .cxxirc file will be created that manges the location of the COMBATXXI binary, JDK, and RTI file that assists with the High Level Architecture capability of COMBATXXI.[26] The user is referred to the COMBATXXI user's guide for detailed installation instructions.[27] At the time of this report, we used the stable build dated 20141027 with Java Development Kit 1.7.0_67.

[24]TRADOC Analysis Center-White Sands Missile Range. *COMBATXXI Wiki*. 2014. URL: `https://cxxi.wsmr.army.mil/Welcome.html`.
[25]Idem, *COMBATXXI Users Guide*, op. cit., COMBATXXXI Install.
[26]Ibid., COMBATXXI Setup Tool.
[27]Ibid., COMBATXXI Install.

## COMBATXXI Scenario General Information

A COMBATXXI scenario consists of six major components: terrain data, entities, performance database, force structure, orders, and entity behaviors. The COMBATXXI user's guide[28] describes all of these elements in detail, however we will highlight aspects relevant to integrating COMBATXXI with EASE. It is also important to note that in the normal context of using COMBATXXI as part of a study, the time to complete the setup of a COMBATXXI scenario can take a few months to finish.

### Terrain

COMBATXXI has very robust capabilities in integrating terrain features such as land, water, building, vegetation and elevation along with the ability to import files from programs such as ArcGIS, Global Mapper, Java, and C++.[29] The difficulty for the user will be ensuring that the entities behave in a desired manner while navigating a specific terrain feature.

### Entities

Entities are the agents that interact with each other in the COMBATXXI scenario. These include individual Soldiers, ground vehicles, aircraft, sensors, etc. A key aspect of developing entities in COMBATXXI is ensuring the entities' profiles are configured according to the requirements of the scenario.

### Performance Database

COMBATXXI contains seven different performance and configuration databases that are configured in Army Material Systems Analysis Activity (AMSAA)'s Standard File Format (SFF). The seven databases are:

- Performance and Characteristics data database.
- Model configuration database.
- Mobility database.
- Communications database.
- JCOMBIC database (obscurant generation parameters).
- Space database.
- Operations database.

The performance database consists of 258 tables of data for various physical and performance characteristics of agents used in a created scenario.[30] The model configuration database contains data that the model uses to create entities and interpret vulnerability data.[31] The

[28]Ibid., Scenario Integration Tool Suite (SITS).
[29]Ibid.
[30]Ibid., Data Architecture and Concepts.
[31]Ibid., Creating an Entity.

mobility database entails information about driver, plow, terrain, vehicle characteristics, vehicle climate zone speeds, and vehicle mobility.[32] The core model uses the data in the communications database to configure the communication network in the scenario, and the JCOMBIC database models smoke and other battlefield obscurants.[33] Finally, the space database is for orbiting satellites, and the operations database contains data the model uses to execute orders and maneuvers in the scenario.[34]

Once a user develops a scenario in SITS, the model loads on the data that is necessary to execute the scenario.[35] Although most of the data for COMBATXXI's performance database comes directly from AMSAA, the developers at Training and Doctrine Command Analysis Center-White Sands Missile Range (TRAC-WSMR) still have to add additional tables in order for the data to work with the model. Therefore, if a user of COMBATXXI plans to significantly modify the performance database, that user should consult with the developers at TRAC-WSMR to ensure the scenario will run as expected. For this project, we did not have to modify the performance database, however the reader is referred to Piotto and Dexter if they modify their data structure to fit COMBATXXI's data structure.[36]

### Force Structure

The user can build units from the team level up to division level. Besides the Soldiers in the units, the user also has the ability to build vehicles, communication networks, sensors, and other assets within the force structure as well.

### Orders

Orders are the scripted actions that entities execute in COMBATXXI. These can range from simple movements to highly complex combined arms actions. In the model, entities execute primitive orders, and units execute a collective of primitive orders called compound orders.[37]

### Behaviors

Without behaviors, an entity will do nothing. All behaviors are developed during the scenario integration process and these can become extremely complex depending on the type of operation the user wants to simulate. COMBATXXI has a behavior scripting language to develop behaviors, and users can also develop more complex behaviors using Python.

[32]Ibid., Scenario Info Editor.
[33]Ibid., Scenario Info Editor.
[34]Ibid., Scenario Info Editor.
[35]Ibid., Creating an Entity.
[36]Julia Piotto and Richard Dexter. *Defining the Structure of the COMBATXXI Databases*. Tech. rep. ADB392433, DODA, X5, AR-015-600, DSTO/LOD, DSTO-GD-0744, DEFENCE SCIENCE and TECHNOLOGY ORGANISATION EDINBURGH (AUSTRALIA) LAND OPERATIONS DIV, Apr. 2013, p. 349.
[37]TRADOC Analysis Center-White Sands Missile Range, *COMBATXXI Users Guide*, op. cit., Orders.

## Knowledge Map Scenario

For this project, we will use a scenario developed by Nesbit et al..[38] This particular scenario involves two UAVs searching multiple target areas, shown in Figure 5, with dynamic updates of target locations to each other. The building of this scenario involved highly complex Python behavior scripts and would present a fair representation of the type of scenario an EASE user could encounter.


**Figure 5. Knowledge Map Scenario from.**[39]

## Batch Script

To capture the customizable parameters a user could tune for a COMBATXXI scenario, we follow the methodology illustrated in Figure 6. As shown in the figure, we capture the customizable parameters from the user during the scheduling process in EASE. Next, we

[38]Peter A Nesbitt et al. *Knowledge Representation for Decision Making Agents.* Tech. rep. ADA589932, XA, TRAC-WSMR*, TRAC-M-TR-13-054, ARMY TRADOC ANALYSIS CENTER MONTEREY CA, July 2013, p. 88.

developed a Python file to create a Windows batch script to simulate an EASE user clicking a go button and running a COMBATXXI scenario from the command line. We will also modify any Python behavior files that have global behavior parameters the user can modify. Finally, EASE will execute the model based on the created batch script. We assume that the scenario is completely developed and uploaded for use in the EASE server. The Python script that creates the batch file will:

- Parse the EASE generated XML file with the study parameters
  - Study
  - Stop time (Seconds)
  - Repetitions
  - Rand Stream
- Find the appropriate .cxxi file
- Write the batch file with the study parameters

The batch file will then execute and the user will be prompted to select the data loggers they want to use for post processing and follow-on analysis.

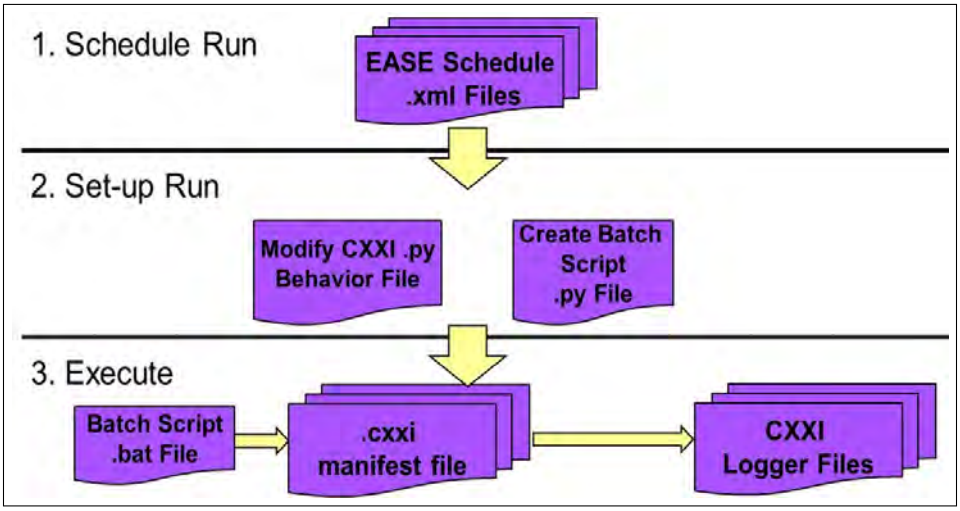



**Figure 6. Methodology to generate COMBATXXI-EASE integration requirements.**

## Data Farming

In this section, we describe a framework for preparing, running, and analyzing multiple variations on a single scenario. In particular this gives the analyst the ability to construct and run a designed experiment, setting certain entities' properties to a pre-chosen set of values. Although COMBATXXI is not usually ran for experimental designs, this methodology is useful during the initial stages of building and testing a model.

### COMBATXXI in Its Current Implementation

The current implementation of COMBATXXI consists of several pieces. First, the SITS program is used to create two important files: the manifest file (whose name ends, by convention, in .cxxi) and the scenario file (an XML file whose name ends in .XML).

The manifest file is small – perhaps a dozen lines – and consists of name-value pairs that describe the locations of certain resources on the computer's file system. For example, one line might show:

```
STD_MOB_DATABASE=databases/StandardMobility
```

to indicate that the "standard mobility" database is located in a folder named "databases" underneath the one that contains the manifest file. The final line of the manifest file names the scenario file associated with the run.

Under the usual practice, the analyst now starts the COMBATXXI graphical user interface. Here the analyst specifies the manifest file (which in turn specifies the scenario file). The analyst also sets certain parameters available at run time, like the "real-time" duration of the scenario or the random number generator, and specifies a configuration file that describes which events are to be logged. Finally the scenario is started with the click of a button. The simulation generates logs of a couple of sorts, including tabular, text-based records of events. When the simulation is complete, the analyst can then import those records into a relational database. The database of choice is HSQLDB, an open-source relational data base management system to which clients can connect by means of JDBC ("Java database connectivity").

Figure 7 shows a schematic of the usual way that COMBATXXI is used. The analyst runs SITS to create the manifest and scenario files (of course this can have been done much earlier). Then the GUI is used to specify the manifest, run-time parameters and the logging configuration, and to run the simulation. Finally, the resulting logs are imported into the database for later analysis.

### A Note on Environmental Variables

Our solution makes broad use of environment variables, so we describe those in some detail in this section. An environment variable is a name and value pair that can be set by one program and accessed by another. Some environment variables are more-or-less permanent, but it is possible for one program to create new ones and for those new values to propagate to a second program called by the first. On Windows, the value of an environment variable is extracted by surrounding the variable's name with % signs; on Linux, where variables are case-sensitive, the value is extracted by preceding the name with a $ sign. So on Windows, one batch file may set a variable with a command like SET ABC=123 and call a second program; that second program can examine whether the variable exists and obtain its value

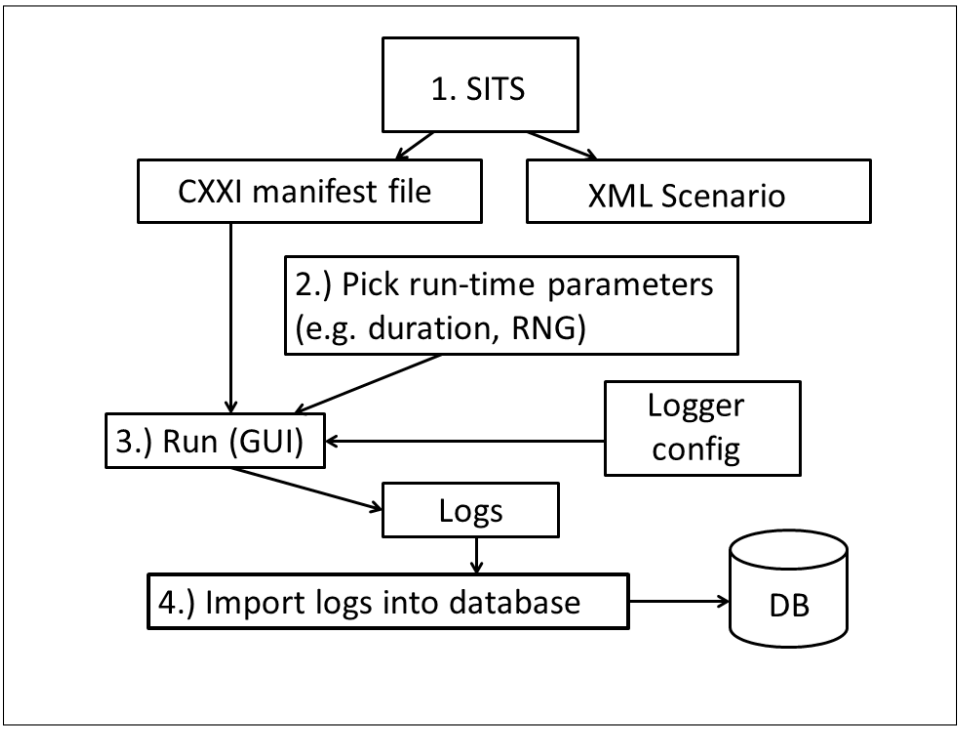



**Figure 7. General COMBATXXI Flow Diagram.**

with %ABC%. Programs like R and Python can examine, create and modify environment variables as well.[40] This gives those programs the ability to communicate program settings to a DOS batch file. For example, the batch file might include this line, which tests for the existence of an environment variable named RANDOMSTREAM and sets it if it is not found (with the caret character, ^, indicating that the line is incomplete):

```
if not defined RANDOMSTREAM ^
(set RANDOMSTREAM=MERSENNE_TWISTER)
```

Each of the environment variables is evaluated at the time the batch file is invoked. If RANDOMSTREAM was set by R, the value of that variable will be the one used in this command. Notice in particular the MANIFEST variable, whose value provides the location of the manifest file. Through this mechanism, the R program can call the same batch file any number of times with different manifest files, random number streams and other settings.

We note that the manifest file can interpret environment variables in some circumstances but not, seemingly, in others. For example this line:

```
COMMS_DATABASE=%INSTALL%/res/databases/commsdata_demo.odb
```

[40]R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2015. URL: `https://www.R-project.org/`; Python Software Foundation. *Python Language Reference, version 2.7*. 2015. URL: `http://www.python.org`.

seems to correctly interpret the value of the environment variable INSTALL. However, the scenario file itself cannot, apparently, be specified as an environment variable.

### The Tools for Data Farming

Data farming is a process that combines simulation models, high performance computing, design of experiments, and statistical analysis to answer research questions of relevance.[41] Since the primary goal of ERS is to analyze multiple systems within a tradespace, using data farming with complex simulation models could provide a way to reduce some of the ambiguities associated with the analyzed tradespace.

**Note:** This document describes an implementation in Windows. While we used Windows 7, other versions should work in the same way. We have experimented with Linux and this scheme seems to work there as well, with some modification, in our limited trials.

In addition to SITS and the COMBATXXI executable, our scheme uses R, an open-source statistical environment that is widely used. It would be straightforward to implement our sceheme in another language like Python; the only requirement is that the tool be able to handle XML and, if needed, JDBC in order to connect to the database.

Our farmer() function in R automates three tasks. First, it reads the farmer input file, which specifies how the different runs of the simulation will differ. Second, it constructs a manifest and scenario file for each run. A single "runner" file, which actually runs the simulation, does not to be modified; instead it is called with differing values of certain environment variables. The third task is to run all the simulations, and to track their progress so as to know when they are all finished. A natural fourth task, not yet implemented, would be to automate the importation of the logs into the database. We have a technical solution for this but it is not yet integrated into the farmer() function. The complete code for the farmer function and runner file can be found in Appendix D. The next sections describe the three tasks performed by the farmer() function.

### Parsing the Farmer Input File

The famer input file contains the instructions that describe how runs differ one from another. Entries can be global, meaning that they apply to all runs, or local, meaning they apply only to individual runs. The format is transactional; that is, different instructions appear on different lines. This makes the file readable. However, there is also the ability to make the format tabular, with all the instructions for one run placed on one line. In this case the required separator is a semi-colon or a tab. Examples of both types of file are given below.

The farmer file may contain comments, which are lines starting with the hash sign, #. There are also two special indicators: Global and Run. We describe the use of these below.

[41] Gary Horne and Ted Meyer. "Data farming and defense applications". In: (2010).

Other than these indicators, there are two types of entry in the file: values and variables (which are the same as "environment variables"). Each of these can be either global or local. Global values specify things like file locations. These appear at the top of the file. Global variable specify run-time arguments like stopping time, random number generator, and configuration log file. These appear after the Global indicator, up to the first Run indicator. Both global values and global variables have a <name>=<value> form, with no spaces around the equals sign. For example, stoptime=86400 is a valid entry that would set the stoptime to be 86400 seconds (24 hours) for every run, unless that setting were over-ridden by a local setting.

The values for each run start with a Run indicator; the indicator itself is followed by the name to be associated with the run. (The name is not required, but the names the system will make up are unattractive, so names are recommended.) Lines after the Run indicator carry either variables, in <name>=<value> format, or values, which have format <id><space><property><space><value>. For example, the line:

```
DEFAULT_T-72_60 Longitude -106
```

specifies that for a particular run, the Longitude property of the object DEFAULT_T-72_60 should have value –106.

**Example**

An example of the farmer function is below. This one produces four runs, named"first","second","third", and"fourth". Of course we recommend more meaningful names. Note that the Windows farmer file uses the Windows path convention, in which directories are separated by the backslash character, \. We have removed comments but recommend that they be included as needed. Notice that the default STOPTIME is set to be 86400, but over-ridden just for Run first. For the second and subsequent runs, the global value is used. Similarly, the third run uses a different random number generator. If a variable is not present in either the global or local sections of the file, a default value is used.

```
#  Example farmer file. This line is a comment.
# These next three lines give the directory holding the
# manifest file and the DOS batch runner.
base=h:\trac\combatxxi\Model\Scenarios\KROE4_scenario
cxxi.0=KROE4_scenario.cxxi
runner=runner1.bat
# "Global" introduces global environment variables.
Global
RANDOMSTREAM=MERSENNE_TWISTER
STOPTIME=86400
LOGFILE=h:\trac\CombatXXI\Farming\logconfig_1.xml
OUTDIR=h:\trac\CombatXXI\Data\TableDump
```

```
# Each "Run" indicator is followed by a name. Then come
# variables and/or value lines for that run.
Run first
STOPTIME=172800
DEFAULT_T-72_60 Longitude -106
DEFAULT_T-72_60 Latitude 34
Run second
DEFAULT_T-72_60 Longitude -106.2
DEFAULT_T-72_60 Latitude 34
Run third
RANDOMSTREAM=SIMKIT_CONGRUENTIAL
DEFAULT_T-72_60 Longitude -106
DEFAULT_T-72_60 Latitude 34.2
Run fourth
DEFAULT_T-72_60 Longitude -106.2
DEFAULT_T-72_60 Latitude 34.2
```

Lines can be combined with semi-colons or tabs; so that last line could have been written Run fourth; DEFAULT_T-72_60 Longitude -106.2; DEFAULT_T-72_60 Latitude 34.2. The farmer()function reads the file and stores it in an R list.

**Constructing the Scenario and Manifest Files**

The example above calls for four runs of the simulation, so the farmer file creates four pairs of manifest and scenario files. The first run has a STOPTIME, or duration of, 172,800 seconds (two days), while the others use the default of one day. Analogously, the third run uses a random number generator different from the other three. All four runs use specific settings for the latitude and longitude of an item called DEFAULT_T-72.

**Running the Simulations**

The simulations are actually run using a file called runner1.bat (see appendix D), which we call the"runner". R sets the global environment variables and then the local ones; then it uses its built-in system() command to invoke the DOC command interpreter, cmd.exe, passing to it the location of the runner. Thus the runner is executed just as it would be at the DOS command line.

The runner examines some environment variables and sets them to defaults if they have not been specified. This step is redundant if runner1.bat is being called from R, since the farmer always sets these defaults; but this mechanism makes is possible to call the runner from DOS as well. Then the simulation is run using the following command:

```
java -cp %CLASSPATH% -Xmx1G cxxi.model.CombatXXIModel ^
CXXIBASE %CXXIBASE% -noview -standalone ^
-stoptime %STOPTIME% ^
```

```
-randomstream %RANDOMSTREAM% -study %MANIFEST% ^
-logs %LOGFILE% -output %OUTDIR%
```

Notice that the manifest file, in particular, is specified in an environment variable. Finally, when the simulation is complete, it deletes a particular temporary file that R created (see appendix D).

It is important to note that the runs are performed in parallel. This is an excellent plan when the number of runs is smaller than the number of cores in the computer's CPU. If there are more runs than cores some degradation of performance would be expected. In that case a small change to the farmer() function could allow the user to specify the number of jobs to be run at once —say, 16 —and to start a second batch when the first batch is complete, using the mechanism that follows.

The mechanism for determining when the simulations are complete is a simple one. Right before the farmer() function dispatches the simulations, it creates a set of temporary files, one for each simulation. The name of this file is passed as the environment variable DELETEME. When a simulation finishes, it deletes its DELETEME file. The farmer() function sleeps in 30-second increments, waking up to check whether any of the DELETEME files exist. When none remain, the simulations are complete. In principle, this file should be deleted even if the simulation crashes (since DOS should resume control of the runner's execution if the java part fails), but, as a backup, the farmer() will quit after 60 minutes of waiting. This number is scenario run-time dependent. The user will probably need to increase run time for more complex scenarios. Figure 8 shows a schematic layout of the data farming set-up.

## Postprocessing

COMBATXXI has pre-installed tools to build a database and example SQL scripts for post-processing. The reader is referred to the COMBATXXI guide for detailed information of the multiple postprocessing tools in COMBATXXI; in this section, we will discuss how to build a database from the logger files and then process the data in R.

### Build Database from Logger Tool

The build database from logger files tool can be opened from the COMBATXXI executable with the following steps: Tools|Data Tools|Import Loggers into Database. Figure 9 shows the inital interface once the tool is opened. The user can either keep the default location for the output database, or browse to another custom location. From this screen the user will then browse to the parent directory of the loggers they want to import. Figure 10 shows the appropriate screenshot of this action. The user will then highlight which repetitions and loggers they want to import. Once COMBATXXI creates the database, the user can select their analysis tool of choice to analyze the data.

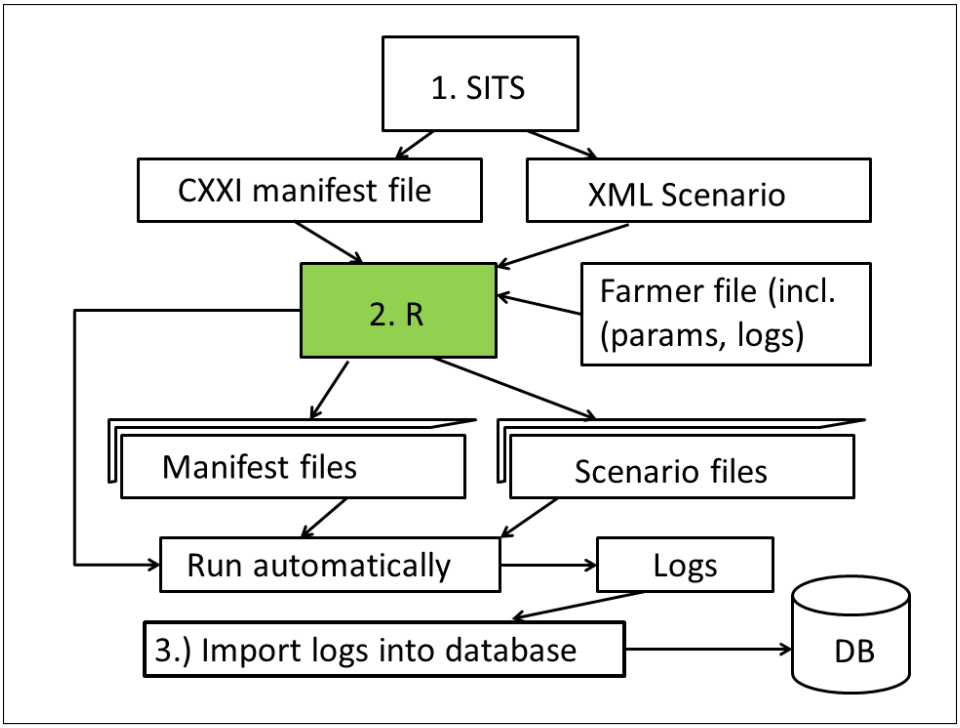



**Figure 8. Schematic of the Data Farming Approach.**

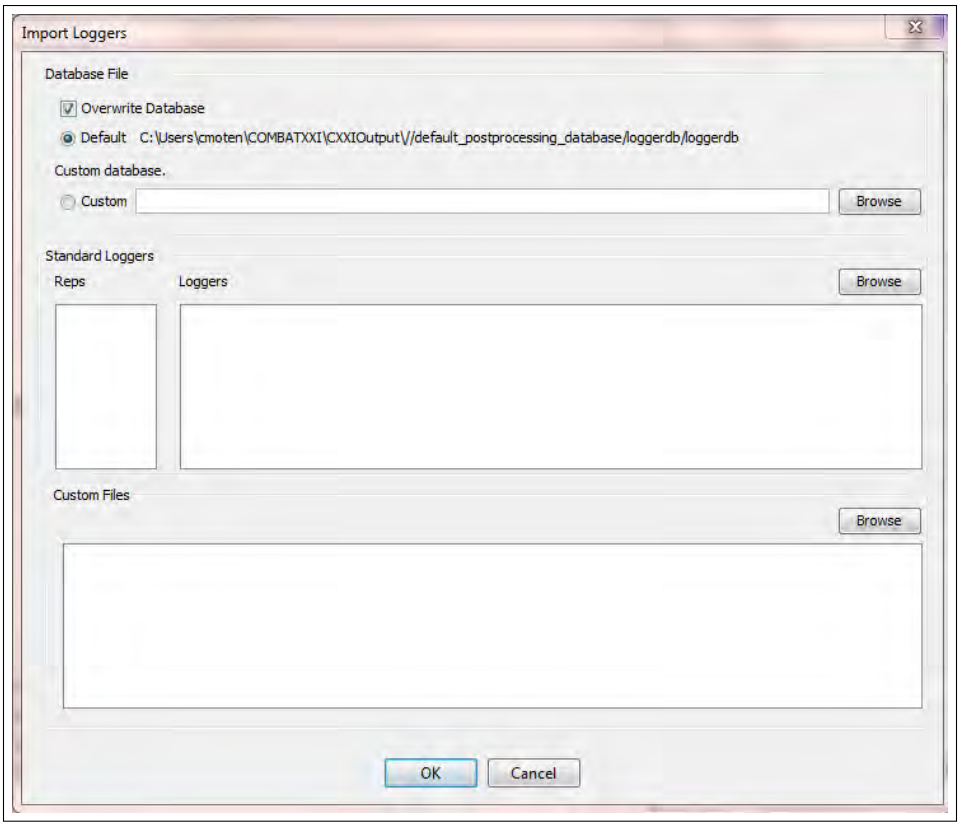



**Figure 9. Import Loggers into Database Initial Interface.**

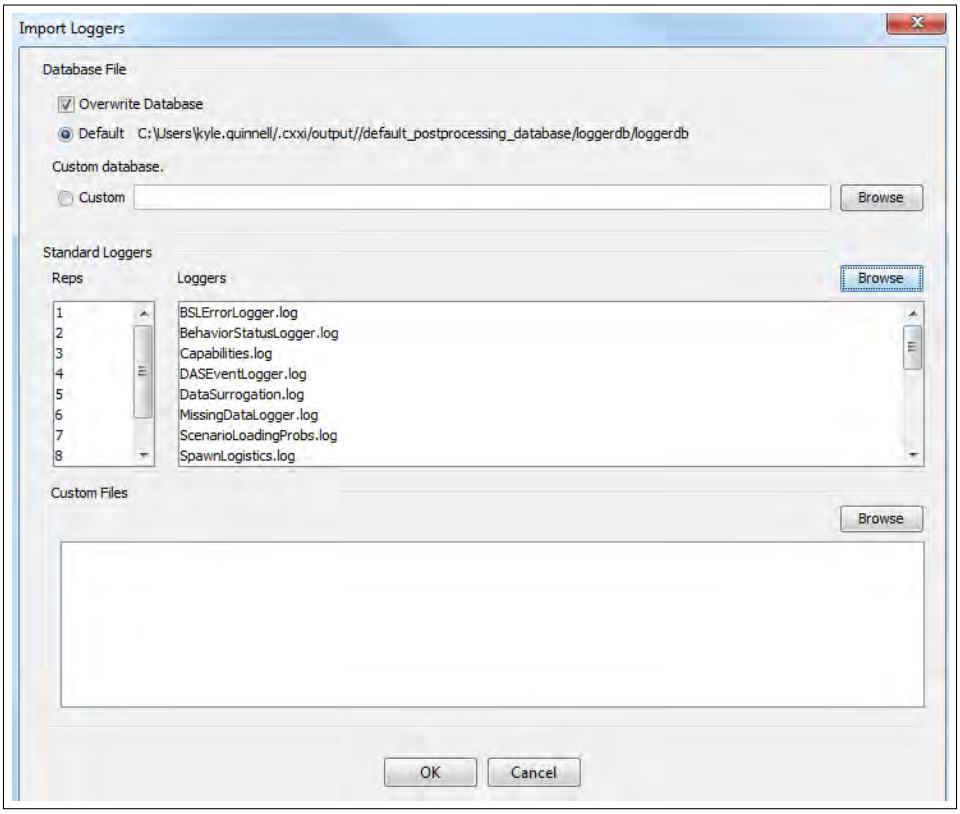



**Figure 10. Import Loggers into Database Reps and Loggers Selection.**

## Accessing COMBATXXI Databases from R and Microsoft Windows

Output from runs of the Combat XXI simulation under Windows are stored in a relational database of the HSQLDB format. This open-source system is the format used in OpenOffice and elsewhere. This document describes how to attach R to an HSQL database, in order to facilitate analysis of the database's contents. This technique has only been tested on 64-bit Windows 7 with R 3.1.2.

### Requirements

### jvm.dll on the PATH

Connection to HSQL is via the ODB package in R. This package relies on the rJava package, and in order to install that package the jvm.dll needs to be available. That is, it needs to on the system and found on the system's PATH environment variable. The dll file ships as part of the JRE, so any computer with the JRE should have jvm.dll as well. If not it will need to be acquired, perhaps from java.com.

The PATH environment variable can be set before launching R from a command-line environment; it can be set globally in Computer | Properties | Advanced System Settings | Environment Variables; or it can bet set within R using a command like this:

```
Sys.setenv (PATH=paste (Sys.getenv ("PATH"),
";C:\\Program Files\\Java\\jdk1.8.0_40\\jre\\bin\\server"))
```

Of course the second argument to paste() is the new piece of the PATH variable being appended to the existing PATH, and it gives the location of the folder where the jvm.dll can be found.

### Required Packages

Once the path is set, the RJDBC package can be installed. The PATH will need to be set whenever the RJDBC package is used. The rJava, and DBI packages are also required. They should be installed along with RJDBC. The ODB package is designed to connect to HSQLDB databases, but is not recommended here.

### JarFile

The RJDBC package requires a "jarFile," a set of byte-compiled Java routines. The Combat XXI database was created using HSQL version 2.3.2, and documentation suggests that a jarFile with the same version is preferred. The version 2.3.2 jarFile can be acquired from http://hsqldb.org/.

### The Database's Name

For the purposes of RJDBC, the database's name is the folder in which the database files are located. (This is different from the convention used by native HSQLDB, where the database's name is the root name of the set of files inside the folder.) The default folder name seems to be default_postprocessing_database/loggerdb, so this name (perhaps qualified by a drive name and path if needed) is the reference name to be used in R.

### Connecting

The new oo() function adapts the odb.connect() function from the ODB package to connect to a database using the name convention described above. The return value is a "handle" to be used in subsequent calls using the DBI interface (see the help for dbListTables, for example).

Here is some sample code showing an example of a successful connection. R is running in a disk location in where there is a database folder and an installation of hsqldb. Those two file locations could, of course, be fully specified. The "base" argument names the base name associated with the database files.

```
# Connect
db <- oo ("default_postprocessing_database/loggerdb",
base = "loggerdb",
jarFile = "hsqldb-2.3.2/hsqldb/lib/hsqldb.jar")
# List all the tables in the database
dbListTables (db)
# The new baseTables object lists the tables that appear in
# an empty database. Other tables are interesting
setdiff (dbListTables (db), baseTables) #
# Read in one of them for processing
olog <- dbReadTable (db, "OBSERVELOGGER")
```

## JDAFS

Appendix B shows the use case that describes the conditions we assume for the use of JDAFS within EASE. To determine the operating requirements and customizable parameters for JDAFS, we took the following steps:

- Install the latest version.
- Build the JDAFS scenario file.
- Develop a batch script to automate the model runs.

## Installation

JDAFS has limited distribution, with only authorized DoD agencies and contractors eligible to receive a copy. Requests should be directed to Director, TRAC-Monterey. JDAFS can execute on 32-bit and 64-bit Windows, as well as on 64-bit Mac OS X and Linux environments. Upon approval, a zip file for the appropriate operating system is downloaded and the contents extracted. The extracted files contain all that is necessary for running JDAFS. On Windows platforms a .bat file is executed, while on Linux and Mac OS X systems a .sh script is used to execute JDAFS. Each platform contains two similar scripts: one for running in GUI mode and another for command-line execution. The appropriate command line script will be used for executing in the EASE architecture.

Compressed files containing all the necessary files for running JDAFS on the most widely used operating systems may be downloaded from `https://eos.nps.edu/jdafs`. Simply uncompress the download to the desired location and execute one of the scripts contained therein. Versions for Windows (32 and 64-bit) as well as Mac (64-bit) and Linux (64-bit) are included. The 64-bit versions are recommended, and the 32-bit Windows version is included for backwards compatibility. The applicable version of a JRE is included in the compressed download files, so they are completely self-contained.

## JDAFS Scenario Creation

This section will briefly describe how to create a JDAFS scenario with an Access database input. The input requirements break down roughly into four areas: run information, platform information, sensor information, and weapons (munitions) information.

### Specifying Run Information

Run information for JDAFS is entered in a table called DAFSScenario[42] and consists of a single row. Rows after the first are ignored. Table 1 shows the column and row information.

**Table 1. DAFSScenario Table.**

| Name | Type | Description |
|---|---|---|
| version | Number | Should always be the value 2. |
| type | String | Typically 'Attack'. |
| bdaFactor | Number | Usually 1 – Probability of correct BDA. |
| replications | Number | Number of replications (1 or more). |

[42]The reason for the name is that JDAFS is built on top of an earlier model simply called DAFS.

### Defining Platforms

Several tables are used to define the basic platforms in JDAFS. The PlatformType table defines each type of platform, and the Movers table defines particular instances of platforms. In addition, a Side table defines the different sides (typically just Red and Blue) and a pair of tables (MoverManager and Waypoint) defines the scripted movement for the platforms. The PlatformType schema is shown in Table 2. Its primary purpose is to identify the types of platforms and to give a value for each. The value is used in the fires allocation algorithm.

**Table 2. JDAFS PlatformType Table.**

| Name | Type | Description |
|---|---|---|
| name | String | Name of platform type. |
| Value | Number | Value of this type of platform. |

Having defined the PlatformType table, specific platforms "Movers") are defined in the Mover table, as shown in Table 3. For simple scenarios, the quantity ("qty") should be specified as 1. If more than one of a specific platform type is desired and the exact location is important only in a broad sense, then qty can be set to more than 1 and the Box table element for that Mover defined in the Box table. The coordinates for locations in JDAFS are somewhat arbitrary, but the distances are in meters.

**Table 3. JDAFS Mover Table.**

| Name | Type | Description |
|---|---|---|
| name | String | Name of Mover. |
| type | String | Type (from PlatformType table). |
| qty | Number | Quantity of this Mover to create. |
| assignment | Text | One of "fires", "sensor", "combined", or "none". |
| affiliation | Text | Blue or Red. |
| xLoc | Number | X coordinate of initial location. |
| yLoc | Number | Y coordinate of initial location. |

Movement for a platform is defined using a MoverManager, which is defined in the MoverManager table, as shown in Table 4. Note that a MoverManager must be defined for every Mover, even if that Mover does not move during the scenario.

A Path Mover Manager is defined by a list of waypoints along which the Mover traverses. The waypoints are defined in the Waypoint table, as shown in Table 5.

If it is desired to create more than one of a particular Mover, their initial locations are defined in the Box table, as shown in Table 6. Each row in the Box table defines an initial rectangular area in which the specified Movers are distributed randomly at the start of the replication. Note that the coordinates are in meters, but are otherwise arbitrary (i.e. the user determines the location of the origin).

**Table 4. JDAFS MoverManager Table.**

| Name | Type | Description |
|---|---|---|
| ID | Number | (optional) Unique ID for this MoverManager. |
| class | String | The (fully qualified) Java name for the MoverManager. Typically dafs.platform.DAFSPathMoverManager. |
| mover | String | The specific Mover this MoverManager is assigned to. If a Mover has qty $> 1$, then a MoverManager is instantiated for each instance. |
| delay | Number | Amount of time to wait before executing the MoverManager's logic; must be $\geq 0$; default is 0. |
| startOnReset | Boolean | If true, then logic starts immediately (subject to delay). Otherwise, wait until another entity signals its start. |

**Table 5. JDAFS Waypoint Table.**

| Name | Type | Description |
|---|---|---|
| mover | String | Specific Mover (as defined in Mover Table) for the waypoint. |
| xLoc | Number | X coordinate of Waypoint. |
| xLoc | Number | Y coordinate of Waypoint. |
| speed | Number | Speed at which the Mover will travel to this Waypoint. If greater than the maximum speed of the Mover, it will travel at only the max speed. |

**Table 6. JDAFS Box Table.**

| Name | Type | Description |
|---|---|---|
| mover | String | Specific Mover (from Mover table). |
| minx | Number | Minimum X coordinate of Box area. |
| maxX | Number | Maximum X coordinate of Box area. |
| minY | Number | Minimum Y coordinate of Box area. |

Sensors are specified in the JDAFS input database in a SensorType table and associated with respective platforms in the Sensor table. The SensorType table contains the data shown in Table 7. The typical "Cookie-Cutter Sensor" is indicated by the class dafs.sensor.DAFSSensor.

**Table 7. JDAFS SensorType Table.**

| Name | Type | Description |
|---|---|---|
| name | String | Name of Sensor. |
| maxRange | Number | Maximum range of sensor. |
| class | String | Fully-qualified class name of sensor. |
| meanTimeToDetect<optional> | Number | For sensors that use a mean time to detect property. |

The sensors are associated with platforms in the Sensor table, as shown in Table 8.

**Table 8. JDAFS Sensor Table.**

| Name | Type | Description |
|---|---|---|
| type | String | The name of the SensorType; refers to a name in the SensorType table. |
| mover | String | The name of the Mover; refers to a name in the MoverTable. |

If the input is an Access database file, the type and mover entries are to be linked to the corresponding columns in the Sensor and Mover tables, respectively. Note that there are more complicated sensor types in JDAFS that are associated with the U.S. Army's Acquire method of detection. These require their own (optional) tables to hold the data required.

### Defining Weapons Effects

JDAFS does not have a graded level of combat damage. Units are either alive or dead. Weapons effects are modeled via a probability of kill for each potential munition-target pairing. This enables the same munition to have different effects on different targets, as well as the same target to react differently to different munitions. Furthermore, the probability of kill can depend on the range of the weapon. The munition types are defined in a MunitionType table and associated with a particular platform in a Munition table. The effects for each munition—target pairing are defined in a LinearKillProbability table. The MunitionType table is shown in Table 9 (Note: for historical reasons, the names of the fields are all uppercase; this will be modified in a future version of JDAFS). The Munition table is similar to the Sensor table in that it associates a MunitionType with a platform.

## Executing JDAFS

JDAFS can be executed either as a stand-alone application with a GUI, or from the command line.

**Table 9. JDAFS MunitionType Table.**

| Name | Type | Description |
|---|---|---|
| MUNITION | String | Name of munition type. |
| WEIGHT | Number | Not used. |
| MER | Number | Not used. |
| MINRANGE | Number | Minimum range of munition (overridden by entries in LinearKillProbability table. |
| MAXRANGE | Number | Maximum range of munition (overridden by entries in LinearKillProbability table. |
| LOAD | Number | Number of "rounds". |
| SPEED | Number | Ground speed of munition. |
| ALGORITHM | String | One of OLD_DAFS, DF, IDF_HE, IDF_ICM. |
| BURST_SIZE | Number | Number of rounds in a burst. |
| SUBMUNITION_COUNT | Number | Number of submunitions in an ICM weapon (not used for others). |

### Executing JDAFS Stand-Alone

JDAFS may be executed in stand-alone mode using its GUI. Before executing substantial scenarios in batch mode (see below), it is recommended that shorter runs be performed with the GUI in order to confirm that there are no errors in the input or other problems that might arise during execution. To run in GUI mode, double-click on the appropriate script file. For illustration, we will be using 64-bit Windows; other platforms are run in a similar manner. For 64-bit Windows, the batch file is called jdafs_win64.bat. When run, a simple map is displayed (see Figure 11). The different menu items allow the user to select an input file and view the execution of the simulation as it occurs.

Running in GUI mode, the user will also be able to identify the version of JDAFS as well as several important libraries, as well as the Java environment. This information can be valuable in tracking down any potential problems due to a mismatch in versions or libraries. The libraries are: DAFS, Simkit (for discrete-event support), Openmap (the GUI), Java itself, LpSolve (for optimizing fires and sensor assignments), Jackcess, and Ucanaccess (for Access database support) – see Figure 12. Clicking on a "More. . . " button on the right side of the panel in Figure 12 reveals more detailed information about the corresponding item on the right. One important piece of information is the operating system and architecture on which JDAFS is being run. Note that the 32-bit version can be run on a 64-bit Windows machine, so this check confirms that the desired one is being executed. As can be seen in Figure 13, the Java version is 1.8.0_11, the architecture is 64-bit, and the operating system is Windows 7.

In GUI mode, the user can select the input file and specify the output file from file chooser dialogs. The scenario is run using the "play" button from the GUI toolbar (see Figure 14). The GUI model typically is used to run a single replication; however, the number of

replications can be specified in the DAFSScenario table (see Table 1). Following execution, the output file must be explicitly saved using the "Save" icon on the toolbar. When the user is satisfied with the small runs in GUI mode, then the long run can be performed in batch mode, as described next.

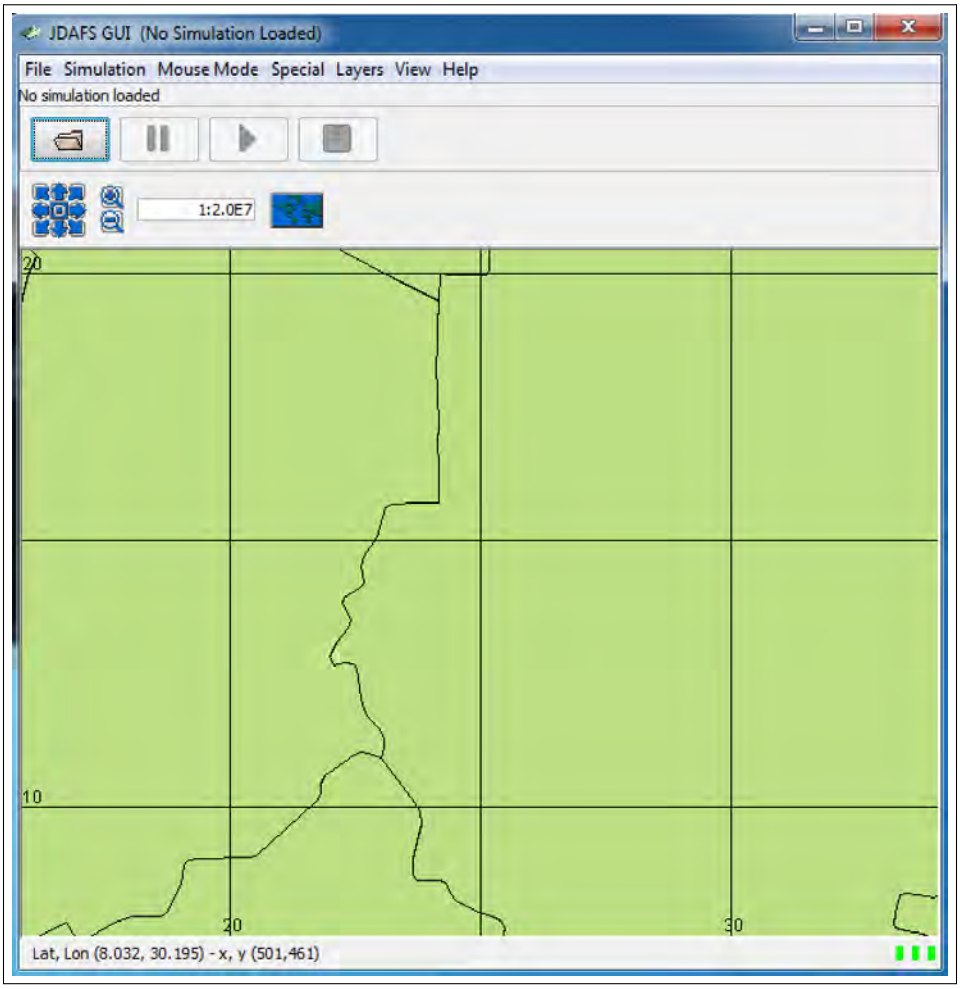



**Figure 11. JDAFS in GUI Mode.**

### Executing JDAFS in Batch Mode

Executing JDAFS in batch mode will be necessary for use in EASE. To that end, scripts containing "_CL" are included in the distribution. Below, examples are given for running JDAFS on a 64-bit Windows platform; executing on the other operating systems proceeds in a similar manner. Here, the script is "jdafs_win64_CL.bat". Each script requires one or two command line arguments. The first argument is the input file, which can be either XML or an Access database file. The second argument, if present, gives the name of the output Access database file for the output tables to be written to. In Figure 15, the output file is SampleScenarioOutput.accdb. JDAFS can also process input files in the "old" Access format (*.mdb). Alternatively, a single argument can be given on the command line, as shown in

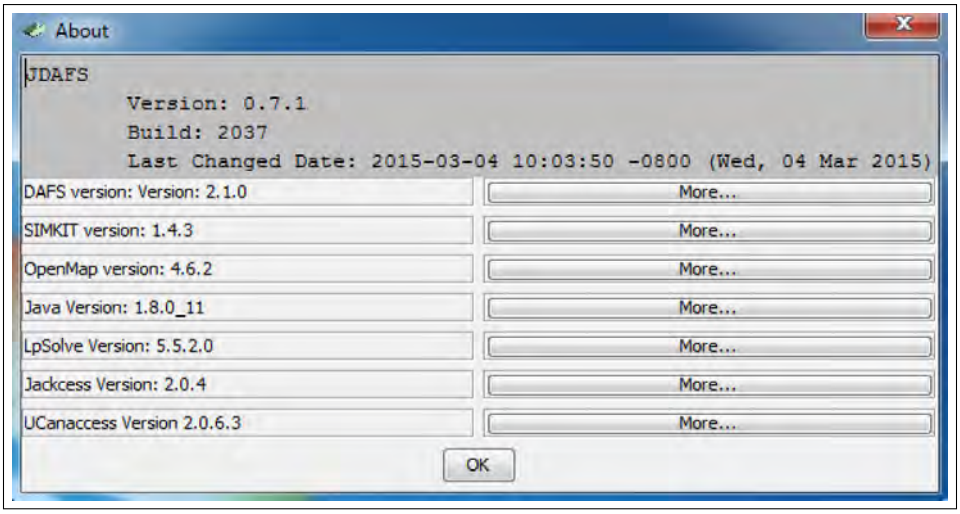



**Figure 12. Help|About Information.**

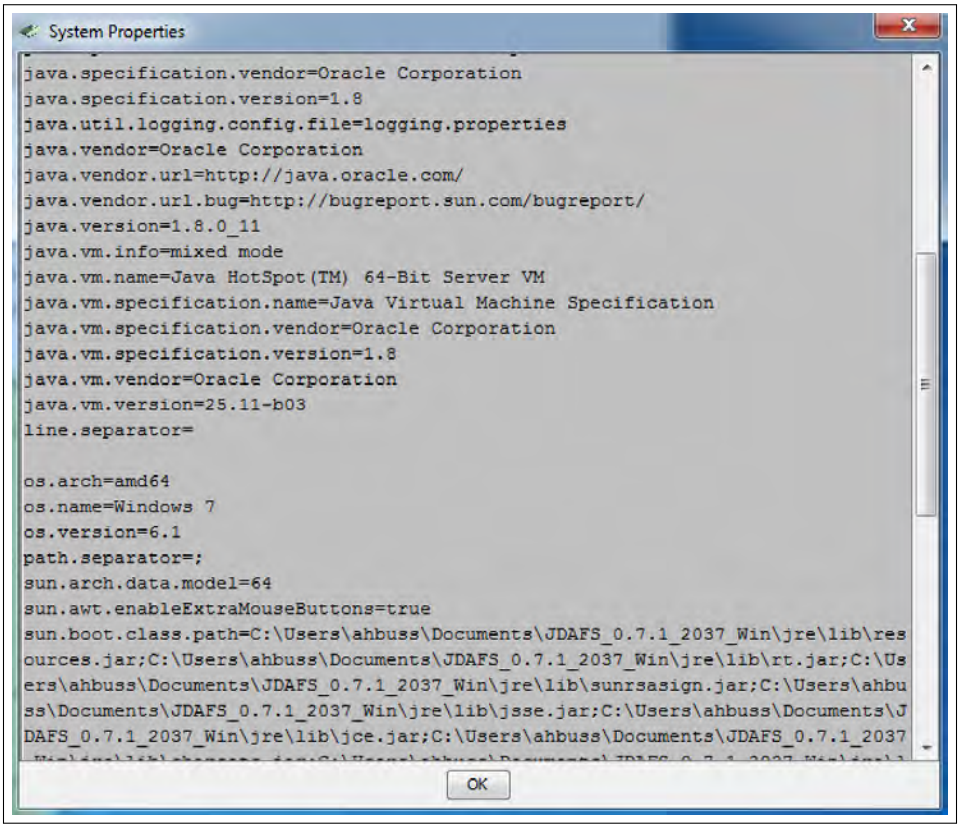



**Figure 13. Some of the Information about the JRE Environment.**

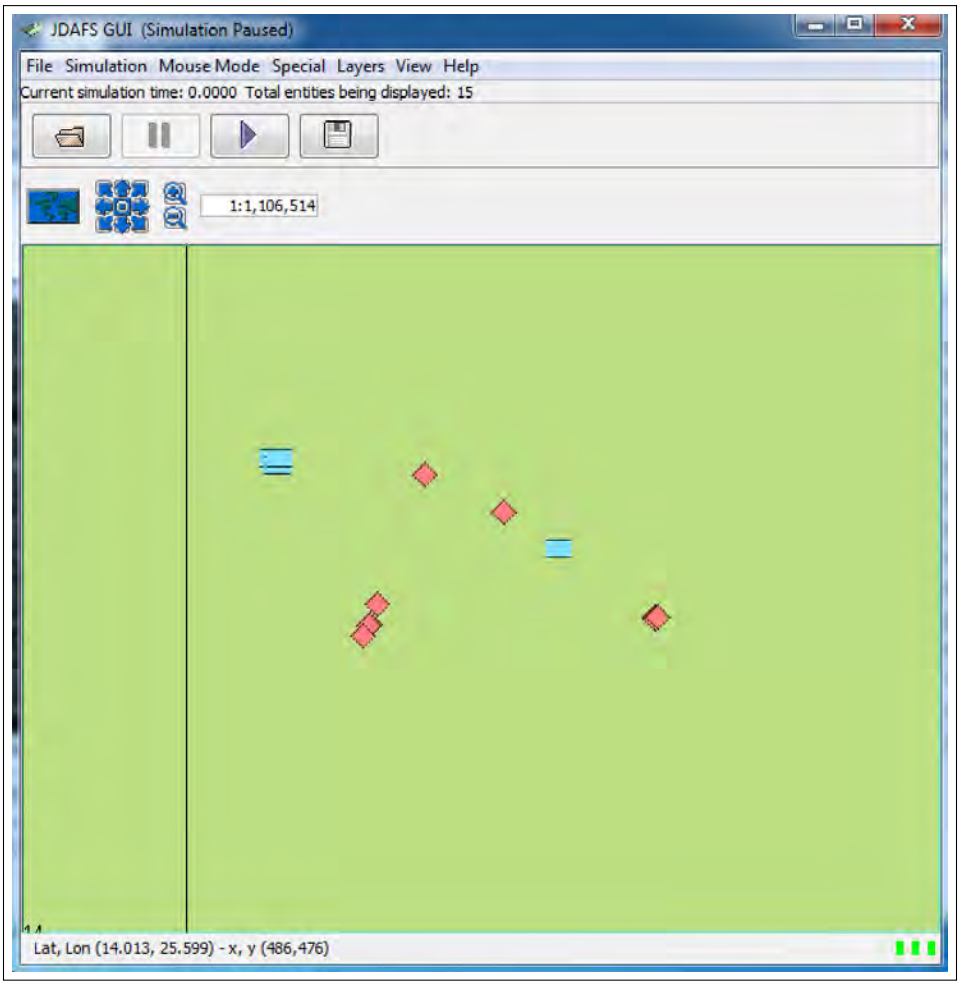



**Figure 14. JDAFS GUI with Simulation Loaded.**

Figure 16. In this case, the output tables are all written to the same input file, overwriting any output tables that might exist.

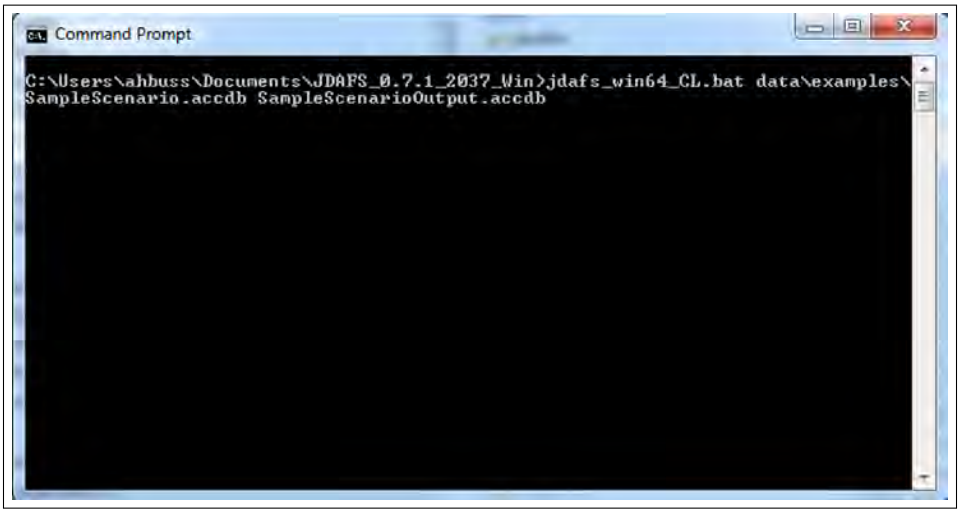



**Figure 15. Executing JDAFS from the Command Prompt (Separate Input & Output Files).**

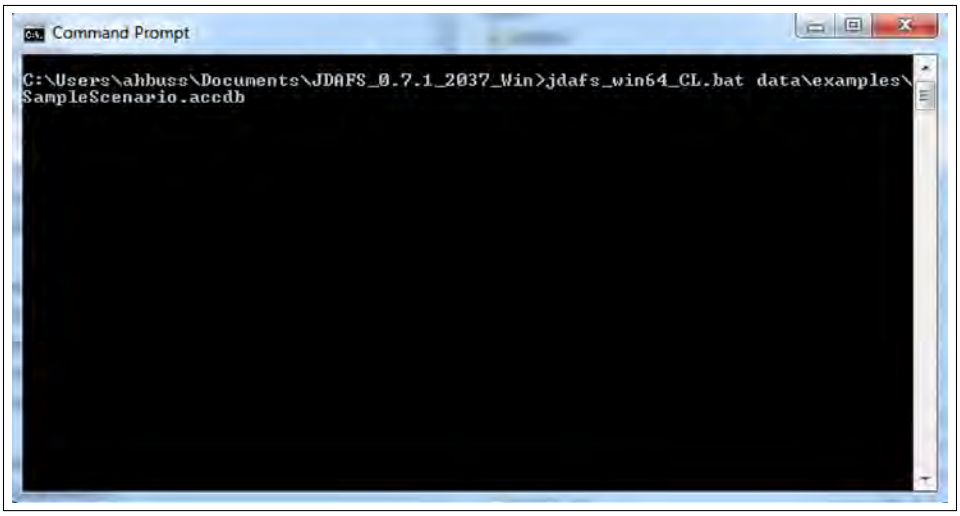



**Figure 16. Executing JDAFS from the Command Prompt (Output Tables Written to Input File).**

### Processing JDAFS Output

The output tables produced by JDAFS are very simple and basic. They typically require more post-processing to perform desired analysis. JDAFS does not explicitly have any specific tools for doing this, since the range of possibilities is far outside the scope of JDAFS minimalism and powerful tools for data processing are readily available. The output consists of four tables: RunInformation, PlatformReport, Acquisition, and KillerVictimScoreboard.

The RunInformation table contains basic information about that specific run of JDAFS (see Table 10).

**Table 10. JDAFS RunInformation Output Table.**

| Name | Type | Description |
|---|---|---|
| DATE_OF_RUN | String | Date of run. |
| INPUT_FILE | String | File used for input. |
| OUTPUT_FILE | String | Output file. |
| VERSION | String | Version of JDAFS used. |

The PlatformReport table, displayed in Table 11, displays information about the final status of each platform at the end of each simulation replication. The data are useful for measures that only depend on the final results of a JDAFS run. The Acquisition table, shown in Table 12, logs all acquisition events, which are either Detections or Undetections. For each such event, information about the sensor and its platform as well as the target and its platform are logged. The killer-victim scoreboard of JDAFS is shown in Table 13. Information about the firing platform, the munition used, and the target platform are all logged, as well as the outcome.

**Table 11. JDAFS PlatformReport Table.**

| Name | Type | Description |
|---|---|---|
| REPLICATION | int | Replication number (1, 2, …). |
| SIMTIME | double | Simulated time at end of replication. |
| NAME | String | Name of platform. |
| LOCATIONX | double | X coordinate of final location. |
| LOCATIONY | double | Y coordinate of final location. |
| VELOCITYX | double | X coordinate of final velocity. |
| VELOCITYY | double | Y coordinate of final velocity. |
| MOVEMENTSTATE | enum | One of stopped or moving. |
| ALTITUDE | double | Typically not used. |
| ASSIGNMENTTYPE | enum | One of fires, sensor, both, or none. |
| BATTLEDIMENSION | enum | Typically not used. |
| DISPOSITION | String | Typically not used. |
| SIDE | String | String (typically Blue or Red). |
| ENTITYVALUE | double | Value of platform, used for fires assignments. |
| ALIVE | boolean | Whether platform is alive of dead at end of replication. |
| COMMANDED | boolean | True if a Commander entity is controlling the platform's movement. |
| TYPE | String | User-specified in the PlatformType table. |

Note that there are three possible outcomes: killed, missed, and killed_again. The first two are obvious, but the third requires some explanation. Since JDAFS models area fires

**Table 12. JDAFS Acquisition Table.**

| Name | Type | Description |
|---|---|---|
| REPLICATION | Int | Replication number (1, 2, …). |
| SIMTIME | double | Time of Acquisition event. |
| EVENT | String | Detection or Undetection. |
| PLATFORM | String | Platform of Sensor. |
| PLATFORMTYPE | String | Platform type. |
| SENSOR | String | Sensor of Acquisition event. |
| SENSORLOCX | double | X coordinate of sensor's location. |
| SENSORLOCY | double | Y coordinate of sensor's location. |
| SENSORMOVING | boolean | True if sensor was moving; false if stationary. |
| TARGET | String | Name of target acquired or lost. |
| TARGETTYPE | String | Type of target. |
| TGTLOCX | double | X coordinate of target. |
| TGTLOCY | double | Y Coordinate of target. |
| TGTMOVING | boolean | True if target was moving; false if stationary. |
| RANGE | double | Distance from sensor to target. |

**Table 13. JDAFS Killer Victim Scoreboard Table.**

| Name | Type | Description |
|---|---|---|
| REPLICATION | Int | Replication number (1, 2, …). |
| SIMTIME | double or String | Time of kill or miss. |
| FIRINGPLATFORM | String | Firing platform name. |
| MUNITION | String | Munition used. |
| LAUNCH_X | double | X coordinate of firing platform. |
| LAUNCH_Y | double | Y coordinate of firing platform. |
| WEIGHT | double | Weight of munition. |
| TARGET | String | Target name. |
| TARGET_X | double | X coordinate of target. |
| TARGET_Y | double | Y coordinate of target. |
| TYPE | String | Type of target. |
| TARGET_VALUE | double | Value of target. |
| TARGET_SIDE | String | Target's side. |
| OUTCOME | String | killed, missed, or killed_again. |

weapons, the logic is such that all enemy (to the firing platform) targets within the blast radius are adjudicated. Since killed platforms are not removed from a scenario, JDAFS adjudicates all platforms for effect regardless of their alive state. Thus, it is possible (and does occur) that a target can be "killed" twice. Since counting second (or subsequent) kills would inflate scores, the "killed_again" outcome is logged in the table under those circumstances.

# LBC

Appendix B shows the use case that describes the conditions we assume for the use of LBC within EASE. To determine the operating requirements and customizable parameters for LBC, we took the following steps:

- Install the latest version.
- Build the LBC scenario file.
- Develop a batch script to automate the model runs.

## LBC Overview

The latest release version of LBC is 3.15.0, and only runs on a 32-bit Windows Java Virtual Machine (JVM). As of this writing, a separate project is underway to update LBC to 64-bit architecture, as well as being able to run on Mac OS and Linux operating systems. The result of that effort will be LBC version 4.x.x. The intent is to provide 100% backwards compatibility with the current LBC version, so that there should not be any major modifications necessary to existing scenario files. The 64-bit version will have several advantages, including faster execution speed and the ability to run substantially larger scenarios than the current version.

LBC can be run either from the command line (using an LBC.bat file that this project created for headless use, such as with EASE) or from a GUI. The LBC GUI is extremely simple (see Figure 17). The two primary options are from the File menu and are File|Run Scenario and File|Convert to XML (see Figure 17).[43]

LBC is executed with a single input file, which can be in one of four formats: Microsoft Excel (either 2007 *.xls or 2010 *.xlsx, *.xlsm, or *.xlsb[44] extensions), Microsoft Access (either 2007 .mdb or 2010 .accdb extensions), OpenOffice database (*.odb extension) or XML (*.xml extension).

LBC output files are plain text, comma-separated value (csv) files. Which files that will be output are determined by a table in the input file (see below). Each output file is written by a specified DataLogger, which outputs particular values of LBC state variables as they change throughout the replications being executed. It is recommended that they all be written to

[43]The Bayes menu item is beyond the scope of this document.

[44]The 64-bit version under development does not support the compressed *.xlsb format.

the same directory. In that case, supplying the end user with the results can be done by zipping the output directory into a single file.

The Run Scenario menu item opens up a File Chooser dialog for the user to select the input file (see Figure 18).[45] Note that selected directory in Figure 18 contains mostly scenarios with MS Excel input files; as noted above, MS Access, OpenOffice database, and raw XML files may also be used. Upon selecting the input scenario, LBC immediately executes the scenario and writes the logged output files to their specified destinations (see description below).

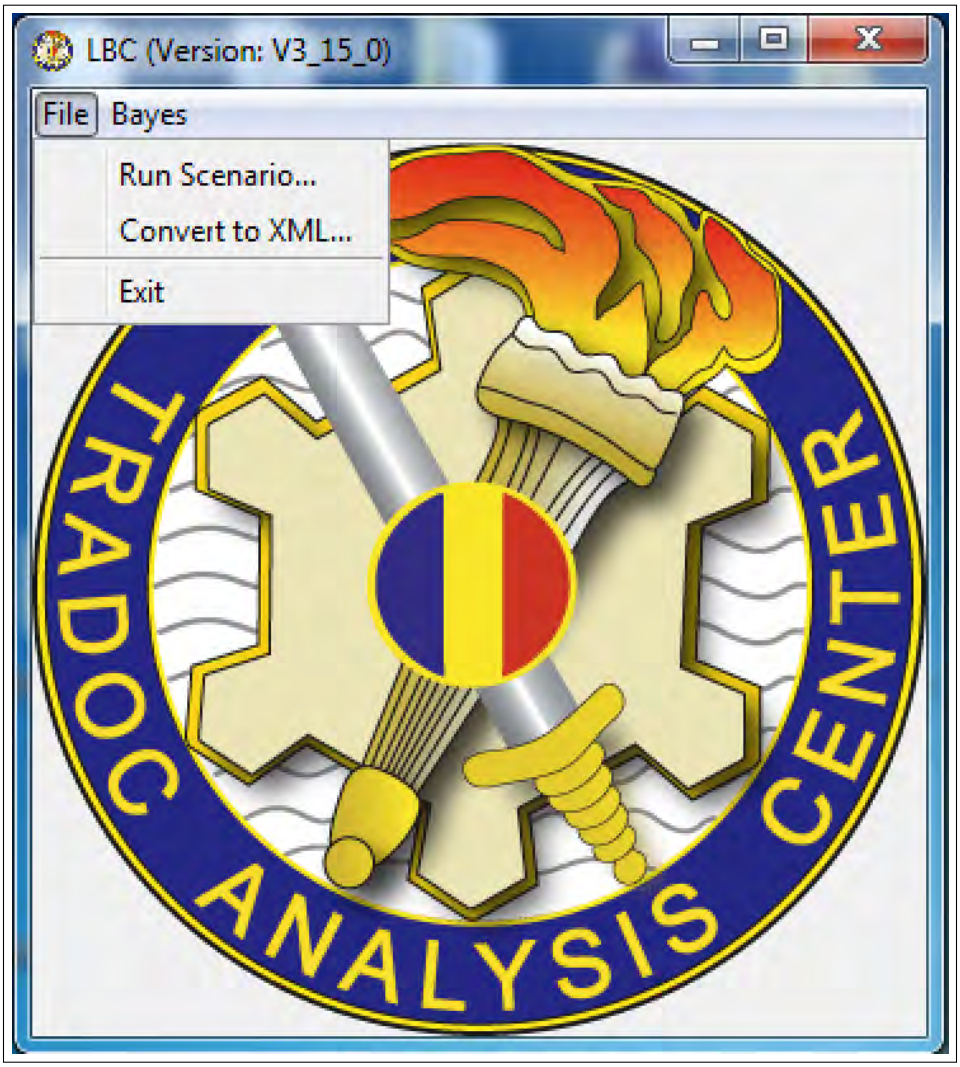



**Figure 17. LBC GUI.**

[45]The Bayes file menu will not be used here.

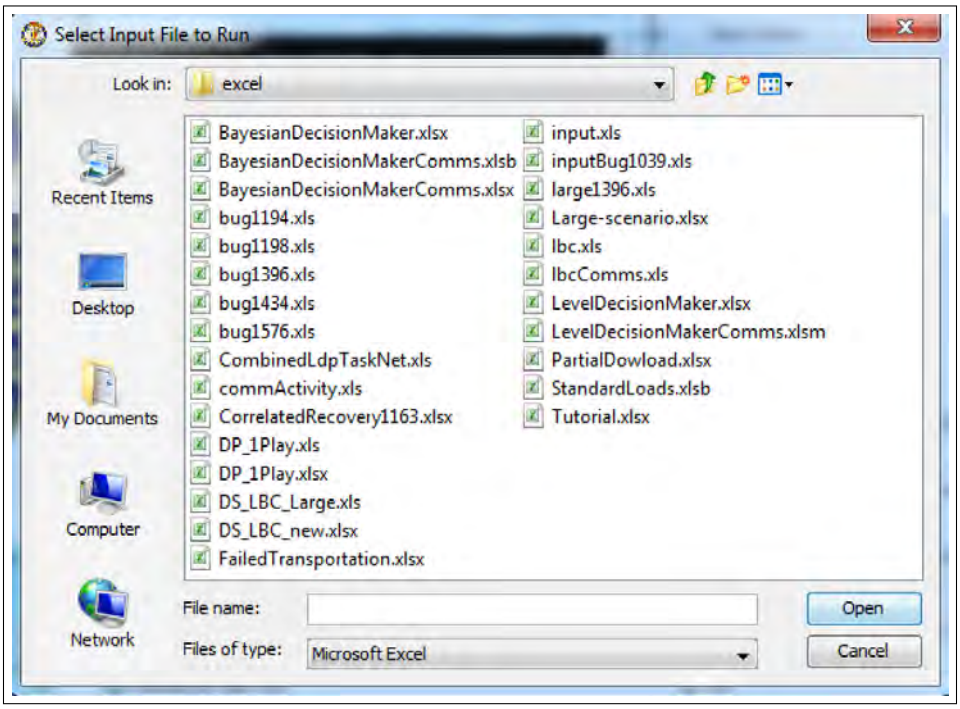


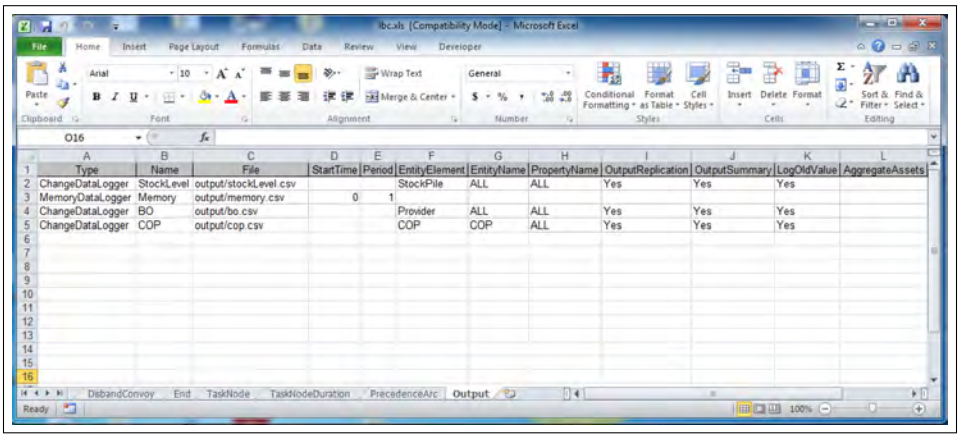

**Figure 18. Selecting LBC Scenario File.**

## Running LBC From the Command Line

In order to run effectively in the EASE architecture, it was necessary to be able to execute LBC from the command prompt. Although LBC already had some of that capability, it was necessary to write a new batch script, LBC.bat, in order to meet EASE operating requirements. The new script executes LBC and then creates a zip file from the outputs (specified in the "Output" table in the input file). For the sake of simplicity, the script assumes that all output files are written to a sub-directory named "output" in the same directory from which the LBC script was executed.

Figure 19 shows a sample Output table in which four output files are specified to be written to the output directory. Note that all the files have a ".csv" extension and that the slashes are forward rather than the typical Windows separators. While the Windows style can be used, the forward slash is preferred and is more platform-independent than the Windows-style slashes. Although LBC Version 3.15.0 only runs on Windows, the newer version under development will run on all Java-supported operating systems.

In Figure 20, LBC is run with the input file lbc.xls, one of the sample scenarios that comes with the LBC distribution. The end of the run is shown in Figure 21. The software used to create the output.zip file is the Open Source "7-zip" program. Figure 22 shows the contents of output.zip after the execution in Figure 21. As noted above, this file can be returned to the user following execution in EASE for further post-processing.



| Type | Name | File | StartTime | Period | EntityElement | EntityName | PropertyName | OutputReplication | OutputSummary | LogOldValue | AggregateAssets |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ChangeDataLogger | StockLevel | output/stockLevel.csv | | | StockPile | ALL | ALL | Yes | Yes | Yes | |
| MemoryDataLogger | Memory | output/memory.csv | 0 | 1 | | | | | | | |
| ChangeDataLogger | BO | output/bo.csv | | | Provider | ALL | ALL | Yes | Yes | Yes | |
| ChangeDataLogger | COP | output/cop.csv | | | COP | COP | ALL | Yes | Yes | Yes | |

**Figure 19. Output Table Specifying Output Files and Directory.**

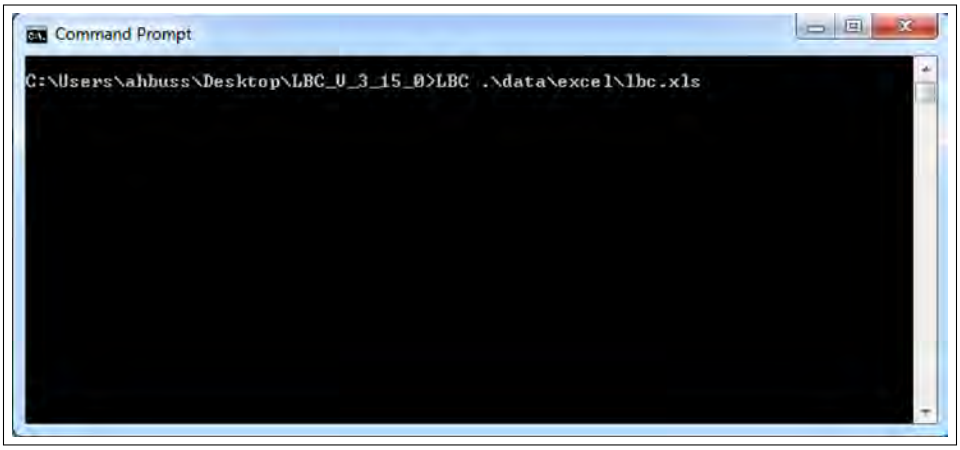



**Figure 20. Executing LBC from the Command Line.**

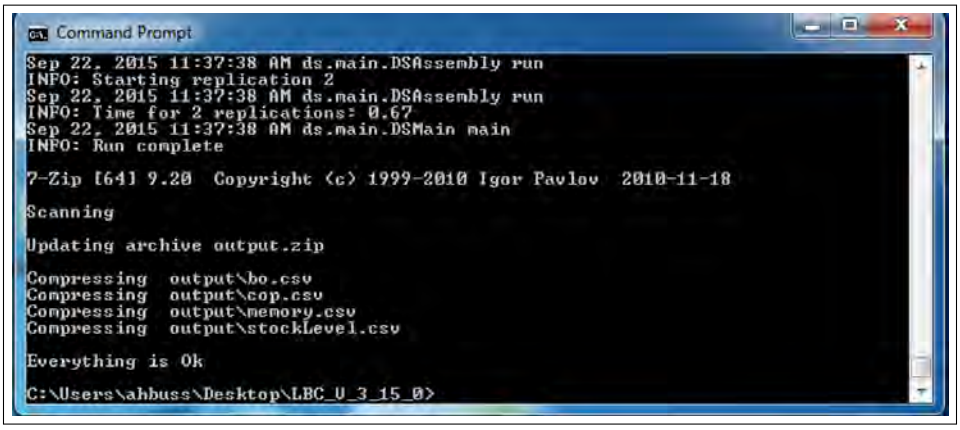



**Figure 21. End of Command Line Execution.**

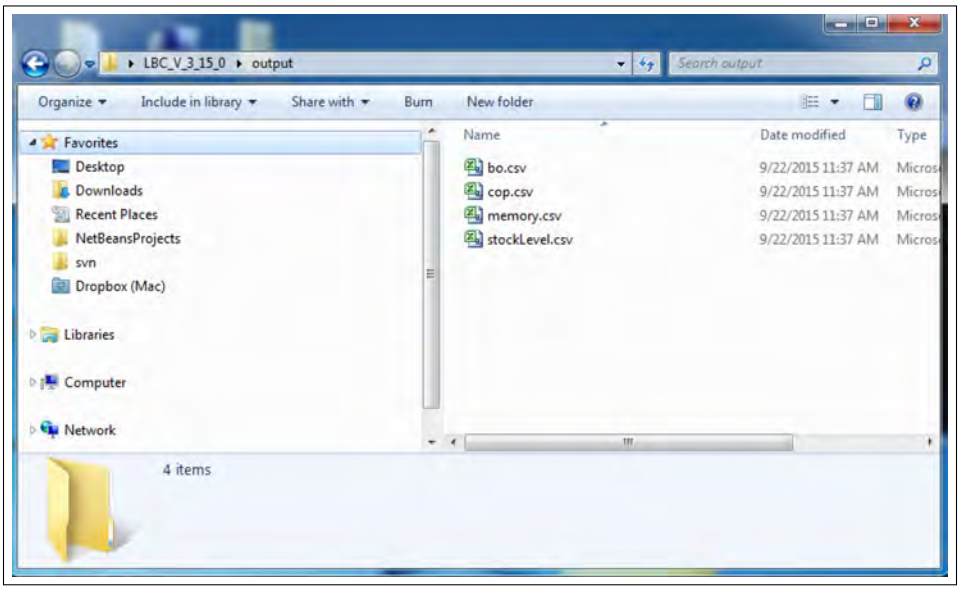



**Figure 22. Contents of output.zip After Execution.**

## Creating a LBC Scenario

A Logistics Battle Command scenario is defined by a single input file, which can be a Microsoft Excel spreadsheet, a Microsoft Access database file, or an Open Office database file. The scenario file can also be in XML format, which is in fact what LBC actually reads to execute a scenario. However, experience has shown that one of the first three input formats is more conducive for scenario creation and modification than editing raw XML. LBC processes an input file of one of the first three types by first converting it to XML format and then reading the generated XML. As a convenience to the user, the generated XML file is saved as a file of the same name but with .xml appended to the file name. The following sections will describe the required tables for creating LBC scenarios. Note that for MS Excel input, a "table" is the same as a "sheet".

### Scenario Information

Information about the scenario's execution is entered in the ScenarioData table (or sheet, if input is Excel) as shown in Table 14.

### Basic Components

### Consumables

The basic logistics element in LBC is a "Consumable". Consumables are defined in the "ConsumableType" table (sheet) (See Table 15). They are not pre-defined, but rather specified by the user and can be arbitrarily named. A Consumable name is typically quite generic (such as "FUEL" or "AMMO". A Consumable may be either integer or floating point (double)

**Table 14. LBC ScenarioData Table.**

| Name | Type | Description |
|---|---|---|
| ScenarioLength | double | How many units of simulated time the scenario will run. |
| Replications | integer | How many independent replications will be executed. |
| Verbose | boolean | If true, then information about each event is logged. |
| reallyVerbose | boolean | If true, then detailed information about each component processing events is logged. |
| ScenarioType | String | Must be one of "LBC" or "DS". |

quantities. For example, FUEL is typically floating point, while AMMO is typically integer. Which type a given Consumable will be is specified in the ConsumableType table.

**Table 15. LBC ConsumableType Table.**

| Name | Type | Description |
|---|---|---|
| name | String | An arbitrary name for the consumable. |
| IntegerQuantity | boolean | If true, then integer; otherwise, floating point. |

### Force Structure

The ForceStructure table, shown in Table 16, lists each unit in the LBC scenario. As with Consumables, the names may be arbitrary, but must remain consistent throughout the scenario. The items that are "not used in LBC" are for use in a Dynamic Maintenance model, which shares code with LBC but is a separate model.

**Table 16. LBC Forcestructure Table.**

| Name | Type | Description |
|---|---|---|
| Unit | String | Arbitrary name, but must be consistent throughout scenario. |
| SystemType | String | Arbitrary name of system associated with the unit that consumes resources. |
| Count | Integer | The number of the SystemType systems associated with the given unit. |
| Priority | Integer | Not used in LBC. |
| WorkRules | Integer | Not used in LBC. |
| WorkRulesLimit | Integer | Not used in LBC. |

### Providers

Providers in LBC provide Consumables, so the SimpleProvider table, Table 17, simply lists the name of each unit. Typically, a given unit can function as both a "provider" and as a "consumer".

**Table 17. LBC SimpleProvider Table.**

| Name | Type | Description |
|---|---|---|
| name | String | Name of simple provider – must match one of the unit names in ForceStructure. |
| Class (optional) | String | Name of the LBC class to instantiate as a Provider/Consumer. If not specified, the default is "lbc.entity.SimpleProvider". |

### Consumption

Consumption in LBC can be represented either periodically or randomly. Periodic consumption allows the user to specify amounts consumed during specified time periods. Random consumption uses probability distributions to define consumption. The data is contained in the PeriodicConsumptionLogic table, Table 18, and is used to instantiate a `PeriodicConsumptionLogic` object. In contrast to the deterministic PeriodicConsumptionLogoc, RandomConsumptionLogic is used for specifying random amounts of consumption. The data for this is specified in the RandomConsumptionLogic table.

**Table 18. LBC PeriodicConsumptionLogic Table**

| Name | Type | Description |
|---|---|---|
| name | String | Name of the PeriodicConsumptionLogic. |
| start | double | Starting time for periodic consumption ($\geq 0$). |
| quantity | double | Amount to consume each period ($> 0$). |
| delay | double | Time between consumptions ($> 0$). |
| consumableType | String | Must correspond to an entry in the ConsumableType table. |

In contrast to the deterministic PeriodicConsumptionLogoc, RandomConsumptionLogic is used for specifying random amounts of consumption. The data for this is specified in the RandomConsumptionLogic table. The Strings in Table 19 with an asterisk denote special strings that specify probability distributions. On input, these are used to create corresponding Simkit RandomVariate instances. Examples of such strings include Normal(500, 50), Gamma(10, 5), Exponential(200), Constant(1), etc. The SimpleProviderConsumables table, Table 20 associates each simple provider (in Table 17) with a consumableType (in Table 15) and a consumption plan (in Table 18 or Table 19).

LevelDemandCommanders are responsible for initiating supply requirements based on the level of a Consumable at a Consumer. The LevelDemandCommander table, Table 21, simply assigns a name to plans and commanders, which are used by other tables. The LevelDemandConsumers table and LevelDemandPlan table, Tables 22 and 23, are supporting tables for the LevelDemandCommander Table.

**Table 19. LBC RandomConsumptionLogic Table.**

| Name | Type | Description |
|---|---|---|
| name | String | Name of the PeriodicConsumptionLogic. |
| start | String* | Probability distribution for starting time for random consumption. |
| amount | String* | Probability distribution for amount to consume each period. |
| nominalInterval | String* | Probability distribution for the random time between consumptions. |
| timeVariability | String* | Currently not used. |
| consumableType | String | Must correspond to an entry in the ConsumableType table. |

**Table 20. LBC SimpleProviderConsumables Table.**

| Name | Type | Description |
|---|---|---|
| providerName | String | Name from SimpleProviders table. |
| consumableType | String | Must correspond to an entry in the ConsumableType table. |
| initialQuantity | double | Initial amount. |
| consumptionLogic | String | Either blank (if only a provider, not a consumer) or corresponds to a name in one of the consumption logic tables. |

**Table 21. LBC LevelDemandCommander Table.**

| Name | Type | Description |
|---|---|---|
| name | String | Defines name of a Level Demand Commander. |
| logisticsPlan | String | Defines name of the logistics plan. |
| logistcsC2 | String | Defines name of a C2 node. |

**Table 22. LBC LevelDemandConsumers Table.**

| Name | Type | Description |
|---|---|---|
| commanderName | String | Name of a Level Demand Commander (from LevelDemandCommander table). |
| consumerName | String | From Providers table. |

**Table 23. LBC LevelDemandPlan Table.**

| Name | Type | Description |
|---|---|---|
| consumer | String | Name of Consumer (from Providers table). |
| consumableType | String | Name of consumable type (from ConsumableType table). |
| desired | number | Consumer's desired amount (integer or double depending on consumableType). |
| order | number | Trigger quantity to request an order. The order quantity is the different between this value and the desired quantity. amount (integer or double depending on consumableType). |
| priorityOrder | integer | |
| minimumForSupply | number | |
| eoq | number | Not used. |

The SimpleLogisticsC2 plans themselves are associated with Consumers and their respective ConsumableTypes, defining a C2 Provider node (see Table 24). The SimpleProviderC2 table, Table 25, defines the ProviderC2s used to fill orders.

**Table 24. LBC SimpleLogisticsC2Plan Table.**

| Name | Type | Description |
|---|---|---|
| index | integer | Increasing integer from 1 to order the entries. |
| name | String | logisticsPlan defined in SimpleLogisticsC2 table. |
| consumableType | String | Defined name ConsumableType, or "ANY" to specify any ConsumableType for the corresponding Consumer. |
| consumerName | String | From ForceStructure table. |
| priority | String | The order priority for this entry; should always be "ANY". |
| providerC2 | String | Defined name of C2 Provider (from the SingleProviderC2 table). |

**Table 25. LBC SimpleProviderC2 Table.**

| Name | Type | Description |
|---|---|---|
| name | String | Defines the name of a SingleProviderC2 . |
| providerName | String | Defines the name of the LogisticsC2 to whom the requirements will be passed. |

### Transportation

There are two ways that LBC models the transport of supplies: using probabilistic delays and explicitly modeling a transportation network.

### Probabilistic Delay Approach

With this approach, there is no specific logistics network defined. The lead time between when supplies are transported and when they arrive is specified by probability distributions. Three tables define this approach. The RandomTransportationDelay table, Table 26, defines the probability distributions for the time it takes to get supplies from one destination to another. Next, the RandomTransportationArrivalProb table, Table 27, defines the probability that a shipment arrives at its destination. Note that a single row with "ANY ANY1.0" means that every shipment does arrive at its destination. Finally, the RandomTransportationNotifyDelay table, Table 28, defines the probability distribution for the delay time for notifying a ProviderC2 that a shipment will not arrive.

**Table 26. LBC RandomTransportationDelay Table.**

| Name | Type | Description |
|---|---|---|
| Index | integer | Increasing, distinct values to order the rows. |
| source | String | The name of the source of the shipment (can be "ANY"). |
| destination | String | The name of the destination of the shipment (can be "ANY"). |
| class | String | The name of the probability distribution of the random delay. Possible values include "Constant," "Exponential," "Gamma," "Triangle," or "TimeVaryingExponential". |
| param1 | double | First parameter of probability distribution specified in class column. |
| param2 | double | Second parameter of probability distribution specified in class column, if the distribution has one. |
| param3 | double | Third parameter of probability distribution specified in class column, if the distribution has one. |

**Table 27. LBC RandomTransportationArrivalProb Table.**

| Name | Type | Description |
|---|---|---|
| Index | integer | Increasing numbers to order rows. |
| source | String | he name of the source of the shipment; can be "ANY" . |
| destination | String | The name of the destination of the shipment; cane be "ANY". |
| probability | double | The probability that a shipment sent from the source arrives at the destination. Must be between 0.0 and 1.0. |

### Transportation Network

A more detailed model of transportation for LBC utilizes an explicit transportation network, consisting of nodes, representing locations of consumers and suppliers, and arcs, over which the supplies will be transported. The delay on each arc is explicitly modeled which, in addition to the topology of the transportation network, provides a more realistic way of modeling the transport of supplies. Substantially more data is required with this approach.

The TransportMode table, Table 29, defines the possible modes of transportation. A common name for use in modeling convoys is "Road". However, like other LBC definitions, the name

**Table 28. LBC RandomTransportationNotifyDelay Table.**

| Name | Type | Description |
|---|---|---|
| Index | integer | Increasing numbers to order rows. |
| source | String | he name of the source of the shipment; can be "ANY" . |
| destination | String | The name of the destination of the shipment; cane be "ANY". |
| class | double | The name of the probability distribution of the random delay. |
| param1 | double | First parameter of probability distribution specified in class column. |
| param2 | double | Second parameter of probability distribution specified in class column, if the distribution has one. |
| param3 | double | Third parameter of probability distribution specified in class column, if the distribution has one. |

itself is arbitrary. The Site table, Table 30, defines a geographic location for the location of Nodes. The network is specified by a collection of Nodes and Arcs. Each row of the Node table, Table 31, corresponds to an instance of a `Node` object in the LBC scenario.

**Table 29. LBC TransportMode Table.**

| Name | Type | Description |
|---|---|---|
| name | String | Defines the name of TransportMode. |

**Table 30. LBC Site Table.**

| Name | Type | Description |
|---|---|---|
| name | String | Defines the name of the Site. |
| latitude | double | The latitude of the Site in decimal degrees. |
| longitude | double | The longitude of the Site in decimal degrees. |

The network connections which connect Nodes are defined in the Arc table. Similar to Nodes, each row in the Arc table, Table 32, represents an instance of an `Arc` object in the LBC scenario. Arcs are directional. The time to traverse an Arc is determined by either the DelayDistribution (if defined) or by the length and speed limit. The SystemTypeData table, Table 33, specifies speed limits for the SystemTypes. The NetworkFailureNotifyDelay table, Table 34, specifies the probability distribution for the delay in notifying the ProviderC2 that a shipment will not arrive. There are also several optional tables for the network transportation model (see Table 35).

### Defining Probability Distributions

There are two ways in which probability distributions are specified in the LBC input: (1) A String consisting of the name of the distribution with the parameters; (2) The name of the distribution with separate numerical columns for the parameters. The first form is shown

**Table 31. LBC Node Table.**

| Name | Type | Description |
|---|---|---|
| name | String | Defines the name of the Node. |
| Site | String | The name of the Site where the Node is located, from the Site table. |
| mode | String | The TransportMode that this Node supports, from the TransportNode table. |

**Table 32. LBC Arc Table.**

| Name | Type | Description |
|---|---|---|
| name | String | Defines the name of the Arc. |
| fromNode | String | The name of the Node at the tail of the Arc; from the Node table. |
| toNode | String | The name of the Node at the head of the Arc; from the Node table. |
| DelayDistribution | String | (Optional) The name of a probability distribution representing the amount of time necessary to traverse the given Arc. |
| speedLimit | double | (Optional) The speed limit for this arc; defaults to infinity. This is ignored if DelayDistribution is present. |
| length | double | (Optional) The length of the given Arc; defaults to zero. This is ignored if DelayDistribution is present. |

**Table 33. LBC SystemTypeData Table.**

| Name | Type | Description |
|---|---|---|
| SystemType | String | Name of the System type (from ForceStructure table) . |
| maxSpeed | String | Maximum speed of system type. |
| mode | String | Transport mode for the type (from TransportMode table). |
| SpeedDistribution | String | (can be blank) The name of a probability distribution representing the amount of time necessary to traverse the given Arc. |

**Table 34. LBC NetworkFailureNotifyDelay Table.**

| Name | Type | Description |
|---|---|---|
| Index | integer | Increasing integers to keep rows in order. |
| source | String | Source of the shipment; can be “ANY”. |
| destination | String | Destination of the shipment; can be “ANY”. |
| class | String | Name of probability distribution for delay. |
| param1 | double | First parameter of probability distribution. |
| param2 | double | Second parameter of probability distribution, if required. |
| param3 | double | Third parameter of probability distribution, if required. |

**Table 35. LBC Optional Tables for Transportation Network.**

| Optional Table | Purpose | Description |
|---|---|---|
| ArcSystemTypeData | Provides additional speed limits based on SystemType for specific Arcs. | Increasing integers to keep rows in order. |
| ArcUnitData | Provides additional speed limits for Units on specific arcs. | Source of the shipment; can be "ANY". |

in Table 36. The other form is shown in Table 37. When defined in this manner, the name column is a String of one of the given names and the param1, param2, and param3 columns are double or numeric values. Note that only the necessary number of parameters for the given distribution are needed.

**Table 36. LBC Possible Random Variates Defined by Strings.**

| String | Example |
|---|---|
| Constant(x) | Constant(1.0) |
| Exponential(x) | Exponential(3.2) |
| Gamma(x, y) | Gamma(2.3, 5.6) |
| Normal(x, y) | Normal(100.0, 10.0) |
| Uniform(x, y) | Uniform(20.0, 50.0) |
| Triangle(x, y, x) | Triangle(10.0, 30.0, 20.0) |
| Beta(x, y) | Beta(3.4, 2.5) |
| InverseGaussian(x, y) | InverseGaussian(5.0, 1.0) |
| LogNormal(x, y) | LogNormal(13.4, 1.3) |
| Weibull(x, y) | Weibull(3.9, 4.5) |

**Schedule Resupply and Task Networks**

Scheduled resupply is modeled using a task network, consisting of TaskNodes and PrecedenceArcs. TaskNodes can be defined in a number of different tables. The TaskNodeDuration table, Table 38, specifies a probability distribution for the time to complete a TaskNode. Complex distributions may also be specified for"shortDuration" and "longDuration".[46] The Plan table, Table 39, specifies the time a unit spends planning. The FormConvoy table, Table 40, defines a Convoy. The Upload table, Table 41, defines Consumables being loaded into a Convoy.

The Move table, Table 42, defines a Convoy moving from one location to another. Requires a transportation network defined. The Download table, Table 43, defines a Convoy unloading at a location. The DisbandConvoy table, Table 44, adds the systems forming the Convoy to its current location. The SeizeAssets table, Table 45, specifies assets being seized from given Providers. The ReleaseAssets table, Table 46, releases assets seized by a SeizeAssets activity. The End table, Table 47, indicates the end of a Plan. The Play table, Table 48,

[46]Lee, *DM/LBC Users Guide v.3.15.0*, op. cit., p. 25.

**Table 37. LBC Random Variates Defined by Name and Parameters.**

| class | param1 | param2 | param3 |
|---|---|---|---|
| Constant | 1 | | |
| Exponential | 3.2 | | |
| Gamma | 2.3 | 5.6 | |
| Normal | 100 | 10 | |
| Uniform | 20 | 50 | |
| Triangle | 10 | 30 | 20 |
| Beta | 3.4 | 2.5 | |
| InverseGaussian | 5 | 1 | |
| LogNormal | 13.4 | 1.3 | |
| Weibull | 3.9 | 4.5 | |

**Table 38. LBC Task Node Duration Table.**

| Name | Type | Description |
|---|---|---|
| name | String | A (unique) name for the corresponding TaskNode. |
| startNoEarlierThan | double | The time before which the task should not start. |
| requiredStart | double | Should be identical to startNoEarlierThan. |
| durationClass | String | Name of distribution (see Table 37). |
| param1 | double | First parameter of distribution (see Table 37). |
| param2 | double | Second parameter of distribution, if necessary (see Table 37). |
| param3 | double | Third parameter of distribution, if necessary (see Table 37). |

**Table 39. LBC Plan Table.**

| Name | Type | Description |
|---|---|---|
| name | String | A (unique) name for the corresponding TaskNode. |
| providingUnit | String | Name of a Provider. |

**Table 40. LBC FormConvoy Table.**

| Name | Type | Description |
|---|---|---|
| name | String | A (unique) name for the corresponding TaskNode. |
| providingUnit | String | Name of a Provider from which to take assets for Convoy. |
| equivalentSet | | |
| systemType | String | Name of a SystemType from ForceStructure table. |
| quantity | integer | Number of given SystemTypes to include. |
| convoyName | String | Name by which the formed Convoy will be known. |
| Priority (optional) | double | Order in which activities queue for seizing assets (defaults to 0.0). |

**Table 41. LBC Upload Table.**

| Name | Type | Description |
|---|---|---|
| name | String | A (unique) name for the corresponding TaskNode. |
| providingUnit | String | Name of a Provider from which the Consumables are taken. |
| convoyName | String | Name of Convoy (from FormConvoy table). |
| standardLoad | String | Name of a standard load to upload to Convoy |
| consumableType | String | Name of a ConsumableType to load (from ConsumableType table). |
| amountCarried | number | Amount of the given ConsumableType to upload. |

specifies a sequence of TaskNodes or other Plays to be used in place of a TaskNode. The PrecedenceArc table, Table 49, defines the precedence relationships between two TaskNodes. The EndPlay table, Table 50, indicates the end of a Play. The SendMessage table, Table 51 specifies information for a message to be sent signaling the completion of a TaskNode. The handling of the received message is specified by the ReceiveMessage table, Table 52.

**Table 42. LBC Move Table.**

| Name | Type | Description |
|---|---|---|
| name | String | A (unique) name for the corresponding TaskNode. |
| receivingUnit | String | Destination of the move. |
| convoyName | String | Name of Convoy (from FormConvoy table). |

**Table 43. LBC Download Table.**

| Name | Type | Description |
|---|---|---|
| name | String | A (unique) name for the corresponding TaskNode. |
| convoyName | String | Name of Convoy (from FormConvoy table). |
| standardLoad | String | Name of a standard load to upload to Convoy. |
| consumableType | String | Name of a ConsumableType to load (from ConsumableType table). |
| amountCarried | number | Amount of the given ConsumableType to upload. |

### Output

Output files are all comma-separated values (CSV) files that use specific DataLogger objects. They are defined in the Output table, Table 53. The possible DataLogger types are shown in Table 54.

**Other Input Tables** LBC has a number of optional input tables that enable additional functionality for a model. These are described in more detail in the DM/LBC Users Guide.

**Table 44. LBC Disband Convoy Table.**

| Name | Type | Description |
|---|---|---|
| name | String | A (unique) name for the corresponding TaskNode. |
| convoyName | String | Name of Convoy (from FormConvoy table). |

**Table 45. LBC SeizeAssets Table.**

| Name | Type | Description |
|---|---|---|
| name | String | A (unique) name for the corresponding TaskNode. |
| providingUnit | String | Name of a Provider from which the required assets are taken. |
| systemType | String | SystemType of assets to seize. |
| quantity | number | Number of assets to seize. |
| assetSetName | String | Unique name that will be used by ReleaseAssets to identify which assets will be released when they are. |

**Table 46. LBC ReleaseAssets Table.**

| Name | Type | Description |
|---|---|---|
| name | String | A (unique) name for the corresponding TaskNode. |
| assetSetName | String | Name of AssetSet defined in SeizeAssets table. |

**Table 47. LBC End Table.**

| Name | Type | Description |
|---|---|---|
| name | String | A (unique) name for the corresponding TaskNode. |

**Table 48. LBC Play Table.**

| Name | Type | Description |
|---|---|---|
| index | integer | Unique integer to order rows. |
| name | String | A (unique) name for the corresponding Play. |
| arcName | String | A unique name for the Arc. |
| fromNode | String | The name of the TaskNode or Play at the tail of the Arc. |
| toNode | String | The name of the TaskNode or Play at the head of the Arc. |

**Table 49. LBC PrecedenceArc Table.**

| Name | Type | Description |
|---|---|---|
| name | String | The name of the Arc. |
| fromNode | String | The name of the TaskNode or Play at the tail of the Arc. |
| toNode | String | The name of the TaskNode or Play at the head of the Arc. |

**Table 50. LBC EndPlay Table.**

| Name | Type | Description |
|---|---|---|
| name | String | A (unique) name for the corresponding Play. |

**Table 51. LBC SendMessage Table.**

| Name | Type | Description |
|---|---|---|
| name | String | A (unique) name for the corresponding TaskNode. |
| providingUnit | String | Name of the Unit sending the message. |
| receivingUnit | String | Name of the Unit receiving the message. |
| messageName | String | A unique name for the message (may be sent multiple times). |

**Table 52. LBC ReceiveMessage Table.**

| Name | Type | Description |
|---|---|---|
| name | String | A (unique) name for the corresponding TaskNode. |
| receivingUnit | String | Name of the Unit receiving the message. |
| messageName | String | The name of the required message. |

**Table 53. LBC Output Table.**

| Name | Type | Description |
|---|---|---|
| Type | String | Type of DataLogger used for this output. |
| Name | String | Arbitrary name to match outputs with logger definitions. Typically identical to the Type value for clarity. |
| File | String | The name of the file to write data to. Should have ".csv" extension. Can include the name of a directory to write the file to, such as "output/cop.csv". |
| StartTime | double | Time logger starts logging data (Not required by all DataLoggers). |
| Period | double | How often logger logs data (not required by all DataLoggers). |
| EntityElement | String | Type of the entity to log data about (not required by all DataLoggers). |
| EntityName | String | Name of entity to log data; may be "ALL" or a regular expression. (not required by all DataLoggers). |
| PropertyName | String | Property to log data for. Can be "ALL" or a regular expression (Not required by all DataLoggers). |
| OutputReplication | String | TRUE or FALSE; if TRUE then output data for all replications are logged. |
| OutputSummary | String | If TRUE, then output a summary over all replications. |
| LogOldValue | String | If TRUE, previous value is logged at the same time as the current value (Only ChangeDataLogger). |
| AggregateAssets | String | If TRUE, then report all repair assets as one record (MMHDataLogger). |
| AggregateFailures | String | If TRUE then report all failure types as one record (MMHDataLogger). |
| FailureUnit | String | Name of the Unit that is the source of the failed assets; may be "ALL" (MMHDataLogger). |
| RepairCommander | String | Name of the RepairCommander responsible for the repairs; may be "ALL" (MMHDataLogger). |

**Table 54. LBC List of DataLoggers.**

| Name | Description |
|---|---|
| AverageDataLogger | Logs the average value of a state (property) over the given period length, starting each period with the current value of the state. |
| TallyDataLogger | Logs the average value of a state (property) over the given period length, without starting each period with the current value of the state. |
| SummationDataLogger | Logs the total of each observation during the given period. |
| ChangeDataLogger | Logs value of states (properties) each time they change. |
| CurrentValueDataLogger | Logs the current value of the states (properties) at the end of each period. |
| MemoryDataLogger | Logs information about the memory usage and execution time for the run. |
| ReliabilityAbortDataLogger | Logs each time a FailurMode fails and is repaired. |
| TimeAverageDataLogger | Logs the time average value of states (properties) over a given period length. |
| ActivityLogger | Logs start time, stop time, and duration of tasks in the task network. |
| ShipmentDataLogger | Logs information about shipments when they are originated. EntityElement should always be "Provider", LogOldValue should always be FALSE, OutputReplications should always be TRUE, OutputSummary should always be FALSE, and the PropertyName should always be "Shipment". |
| MMHDataLogger | Logs maintenance man-hours (MMH) used for repairs. |

**This page intentionally left blank.**

# Chapter 3
# Results

## COMBATXXI

### Operating Requirements

From reviewing the COMBATXXI documentation and completing the batch script, we determined the following operating requirements:

- **COMBATXXI stable build**
- **Manifest File (.cxxi)**
  - Scenario Databases
  - Scenario Scripts
  - Terrain Data Files
  - Map Data Files
  - Icon Files
  - Scenario.cxxi File
  - Scenario.xml File
  - Scenario.odb File
- **.cxxirc file created in the user's home directory.**
- **.bat file to run the model.**

It is important to note that if the user has a requirement to modify any of the above files, then this must be done in a separate process prior to executing the model.

### Custom Parameters

To execute the model the user can modify the input to the following parameters for a COMBATXXI scenario. EASE would then use these parameters as command line inputs for executing the scenario. The should refer to the COMBATXXI User's Guide[47] for further details of these specific parameters. The items with an asterisk (*) were not used in this study.

- **Study:** the manifest file to use.
- **Logs:** the location of the output logs.
- **Reps:** the number of replications of the model to run.
- **Stoptime:** the scenario stop time in seconds.

[47]TRADOC Analysis Center-White Sands Missile Range, *COMBATXXI Users Guide*, op. cit.

- **Savedstatetime*:** the time the model should save the state of the model run.
- **Runwithsavedstate*:** select the saved state file to begin the scenario.
- **Noview:** run the model without viewing it on the map.
- **Production*:** run the model in production mode.
- **Standalone*:** write the standard out (STDOUT) and standard error (STDERR) files to the logger output.
- **Skipcomms*:** run the model with perfect comms.
- **Profilejython*:** profile the python behaviors at runtime.

# JDAFS

## Operating Requirements

- JDAFS stable build.
  - Note that the JDAFS distribution includes all necessary libraries, JRE for the platform in question, a batch script (JDAFS.bat), and sample scenario input files.
- Scenario input file.
  - The input file is a single Access database written according to the structure of the previous section.
  - Alternatively, the input file can be written in XML format with a schema similar to that of the Access database input.
  - The scenario file may be created from scratch or the user can modify an existing input file.
- Log files.
  - Log files for JDAFS are written to a directory `<user.home>/.dafs/log`, where `<user.home>` is the home directory of the user executing the model.

## Custom Parameters

The main execution parameter to be modified is specifying the name and location of the output file. The location can be done either by an absolute file path or one relative to the execution directory (location of JDAFS.bat). The output is in an Access database format. If an output file is not specified, the output tables are written to the same Access database as the input file. This can be convenient when many scenarios are being executed, since the inputs and outputs are kept together.

# LBC

## Operating Requirements

- LBC stable build.

- Scenario input file.
  - Single Access database.
  - Excel files.
  - The input file can be a modification of previous database or developed from a new database.

## Custom Parameters

- The name and location of the output file executed by:
  - The absolute file path.
  - The relative path to the execution directory.
- If an output file is not specified, the output tables are written to the same Access database as the input file. This can be convenient when many scenarios are being executed, since the inputs and outputs are kept together.

**This page intentionally left blank.**

# Chapter 4
# Summary and Conclusions

## COMBATXXI

### Implementation

The time-line in Table 55 gives an estimate on how long it can take to have COMBATXXI fully functional within EASE. Once connected, the EASE system engineer can setup COMBATXXI to run in its current configuration, or the EASE system engineer can use the provided Python script in Appendix C to execute only the model run of a COMBATXXI scenario uploaded to the EASE server. It is most likely that the current users of COMBATXXI would still remain the primary users of this tool in the EASE environment, and therefore would require minimal training on its use. Since EASE, however, does allow for any user that has access to it to utilize the available tools, any user not familiar with COMBATXXI should train on how to build COMBATXXI scenarios outside of EASE.

### Recommended Use of COMBATXXI

In the context of ERS, COMBATXXI is best utilized for studies that either have a long time horizon, or the studied scenario is best analyzed with a high resolution model. In EASE, COMBATXXI should be operated as it is currently configured; this would allow current users to do most of their scenario development outside of EASE, but they could make minor modifications to a scenario once uploaded to EASE.

## JDAFS

### Implementation

The time-line in Table 56 gives an estimate on how long it can take to have JDAFS fully functional within EASE.

### Recommended Use of JDAFS

Because of its minimalistic structure, JDAFS is suited for all modeling and simulations users in the scientific and testing, and analysis communities. JDAFS will allow its users to quickly explore the tradespace and further refine their study scenarios it a higher resolution model is necessary.

**Table 55. COMBATXXI Implementation Timeline.**

| N Date (Months) | Task | Comments | Support Required from TRAC-WSMR |
|---|---|---|---|
| N | Request stable build from Director, TRAC | This requests is handled at HQ, TRAC and would not affect any operations at TRAC-WSMR during processing. | If approved, assist designated personnel with setting up a CXXI Wiki account. |
| N+1 | Receive a copy of COMBATXXI and start training on how the model works. | Potential support could be needed to provide COMBATXXI trainers. | If a training request is approved, based on capacity, provide either mobile training or on-site training to EASE systems engineers. |
| N+3 | Install CXXI stable build | Potential for support could be needed to assist with installation issues. | Assist through JIRA trouble tickets under current business rules. |
| N+4-N+6 | Import CXXI scenarios into EASE | This action is simply uploading files into the EASE server. | None |
| N+7 | Execute the run of a CXXI scenario. | As JCIDS transitions to ERS, the current users of CXXI are expected to be TRAC, MCCDC, TACOM, and other current CXXI users. | Assist through JIRA trouble tickets under current business rules. |

**Table 56. JDAFS Implementation Timeline.**

| N Date (Months) | Task | Comments |
|---|---|---|
| N | Request stable build from Director, TRAC-Monterey | A stable build of JDAFs will be sent to the sponsor as a final deliverable. |
| N+1 | Test JDAFS stable build and run example scenarios. | This is a simple procedure of unzipping the JDAFS file and running the appropriate batch file. |
| N+2 | Import JDAFS scenarios into EASE and run the scenarios to ensure successful connection. | This action is simply uploading files into the EASE server. |

# LBC

## Implementation

The time-line in Table 57 gives an estimate on how long it can take to have LBC fully functional within EASE.

**Table 57. LBC Implementation Timeline.**

| N Date (Months) | Task | Comments |
|---|---|---|
| N | Request stable build from Director, TRAC | This procedure is similar to the request for authorization to use COMBATXXI. |
| N+1 | Test LBC stable build and run example scenarios. | This is a simple procedure of unzipping the LBC file and running the appropriate batch file. |
| N+2 | Import LBC scenarios into EASE and run the scenarios to ensure successful connection. | This action is simply uploading files into the EASE server. |

## Recommended Use of LBC

All members of the ERS community can utilize LBC in their studies. LBC will provide a better understanding of the full operational requirements of an asset within the tradespace because of the inclusion of logistics requirements.

**This page intentionally left blank.**

# Appendix A
# Study Plan

## Problem Statement

The focus of this research was to identify the requirements to integrate COMBATXXI, JDAFS, and LBC into a PaaS cloud computing environment.

### Issues for Analysis

In order to accomplish this purpose, this research addressed the following questions:

**Issue 1:** What are the operating requirements for the simulation models?
- EEA 1.1: What are the operating requirements for COMBATXXI?
- EEA 1.2: What are the operating requirements for JDAFS?
- EEA 1.3: What are the operating requirements for LBC?

**Issue 2:** What are the customizable parameters for the simulation models?
- EEA 2.1: What are the customizable parameters for COMBATXXI?
- EEA 2.2: What are the customizable parameters for JDAFS?
- EEA 2.3: What are the customizable parameters for LBC?

## Project Team


| | |
|---|---|
| **Sponsor Agency:** | Dr. Simon Goerger<br>U.S. Army Corps of Engineers Research and Development Center<br>Vicksburg, MS<br>Simon.R.Goerger@erdc.dren.mil |
| **TRAC Lead:** | Cardy Moten III<br>MAJ, LG/FA49<br>TRADOC Analysis Center - Monterey<br>cardy.moten3.mil@mail.mil |
| **NPS Faculty:** | Dr. Arnie Buss<br>MOVES Institute<br>Naval Postgraduate School, Monterey, CA<br>abuss@nps.edu |
| **NPS Faculty:** | Dr. Sam Buttrey<br>Operations Research Department<br>Naval Postgraduate School, Monterey, CA<br>buttrey@nps.edu |


## Constraints, Limitations, and Assumptions

*Constraints* limit the study team's options to conduct the study. *Limitations* are a study team's inabilities to investigate issues within the sponsor's bounds. *Assumptions* are study-specific statements that are taken as true in the absence of facts.

- Constraints
  - Complete the project by 30 September 2015.
- Limitations
  - The project will focus on one representative use case.
  - The parameters identified requirements depend on the utilized scenario.
- Assumptions
  - The integration requirements between EASE, COMBATXXI, JDAFS, and LBC are feasible to implement.
  - The requirements identified in the use case are sufficient to account for all entities in COMBATXXI, JDAFS, and LBC.

## Methodology

The execution of this project will follow the technical approach outlined in this section. First the team will identify the requirements to install COMBATXXI and execute a COMBATXXI

scenario to include the scenario meta data and customizable parameters. Upon completion of this phase of the project, the project team will apply a similar methodology to implement the connection between JDAFS, LBC and EASE. The project team will explore various methodologies for indexing large databases for faster retrieval of simulation artifacts. Finally, we will document and enclose all efforts in this report. We show the methodology in Figure A–1.

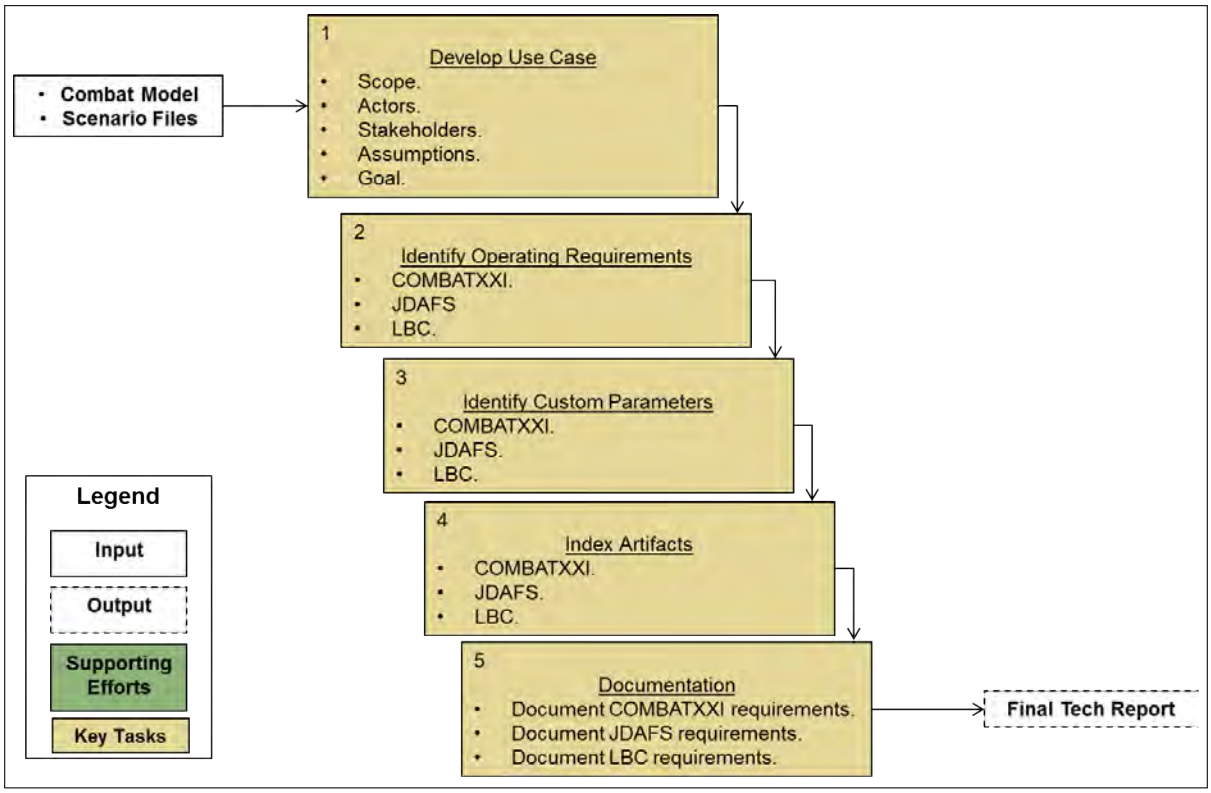



**Figure A–1. EASE integration methodology.**

## Timeline

We executed this project according to the following timeline.

| | |
|---|---|
| **02 SEP 14** | Initial IPR, Review of re-stated problem and approach. |
| **05 JAN 15** | IPR #2, Review of initial results. |
| **15 APR 15** | IPR #3, Review of follow-on results. |
| **30 SEP 15** | Final IPR and final out-brief. |

**This page intentionally left blank.**

# Appendix B
# Use Case Scenarios

## Use Case 1: COMBATXXI-EASE

**Primary Actor:** Study analyst in the ERS community (TRAC,Marine Corps Combat Development Command (MCCDC), etc.)
**Scope:** EASE, COMBATXXI

**Level:** User goal

**Stakeholders and Interests:**

- Study analyst— wants to execute COMBATXXI in EASE
- EASE System Engineer— wants to integrate a COMBATXXI scenario within EASE based on the user's requirements

**Conditions:**

1. The COMBATXXI scenario is available per the user's request.
2. If the scenario is unavailable, the user has the ability to develop and upload the scenario into EASE.

**Success Guarantee:** The user's request is logged and scheduled; the COMBATXXI scenario executes without error; the artifacts are available for analysis in a database.
**Main Success Scenario:**

1. User selects the COMBATXXI scenario to execute from the EASE interview.
2. EASE schedules the execution of the COMBATXXI scenario with the correct customizable parameters provided by the user.
3. The user has the ability to view the execution of the model, if they choose to do so.
4. The artifacts from the model are available for the user to conduct follow-on analysis.

**Extensions:** The user wants to modify either the scenario or the performance data from its current configuration.

1. The user is asked if they want to modify the scenario or entity performance data during the EASE interview.
2. If the user chooses to modify the scenario outside of EASE, the user downloads the appropriate scenario files from EASE.
3. Once the modification is complete, the user submits a request to the EASE system engineer to upload and integrate the modified scenario into EASE.

4. If the user chooses to modify, the scenario within EASE, they should have full access to the appropriate tools to modify and save the COMBATXXI scenario.

## Use Case 2: JDAFS-EASE Novice User

**Primary Actor:** Study analyst in the ERS community (TRAC,MCCDC, etc.)
**Scope:** EASE, JDAFS

**Level:** User goal

**Stakeholders and Interests:**

- Study analyst— wants to execute JDAFS in EASE
- EASE System Engineer— wants to integrate a JDAFS scenario within EASE based on the user's requirements

**Conditions:**

1. The JDAFS scenario is available per the user's request.
2. The user does not have the ability to modify the scenario file.

**Success Guarantee:** The user's request is logged and scheduled; the JDAFS scenario executes without error, and the output database file is available for analysis.
**Main Success Scenario:**

1. User selects the JDAFS scenario to execute from the EASE interview.
2. EASE schedules the execution of the JDAFS scenario with the correct input file provided by the user.
3. The user has the ability to view the execution of the model, if they choose to do so.
4. The output database is available for the user to conduct follow-on analysis.

## Use Case 3: JDAFS-EASE Advanced User

**Primary Actor:** Study analyst in the ERS community (TRAC,MCCDC, etc.)
**Scope:** EASE, JDAFS

**Level:** User goal

**Stakeholders and Interests:**

- Study analyst— wants to execute JDAFS in EASE using a custom-built scenario
- EASE System Engineer— wants to integrate a JDAFS scenario within EASE based on the user's requirements

**Conditions:**

1. The modeler is able to create a JDAFS input file that matched the requirements of the scenario being considered. This would require having Microsoft Access software available to create the database input file or an appropriate XML editor if the input is chosen to be in XML form.
2. The user does not have the ability to modify the scenario file.

**Success Guarantee:** The user's request is logged and scheduled; the JDAFS scenario executes without error, and the output database file is available for analysis.
**Main Success Scenario:**

1. The modeler creates the JDAFS scenario (either Access or XML format) to execute based on the requirements of the analysis being conducted.
2. The modeler executes short test runs locally to ensure the correctness of the input database and correct any discovered errors.
3. EASE schedules the execution of the user-developed JDAFS scenario.
4. The modeler has the ability to view the execution of the model, if they choose to do so.
5. The outputdatabase from the model is available to conduct follow-on analysis.

## LBC-EASE Novice User

**Primary Actor:** Study analyst in the ERS community (TRAC,MCCDC, etc.)
**Scope:** EASE, LBC

**Level:** User goal

**Stakeholders and Interests:**

- Study analyst— wants to execute LBC in EASE
- EASE System Engineer— wants to integrate a LBC scenario within EASE based on the user's requirements

**Conditions:**

1. The LBC scenario is available per the user's request.
2. The user does not have the ability to modify the scenario file.

**Success Guarantee:** The user's request is logged and scheduled, the LBC scenario executes without error, and the output files are available for analysis.
**Main Success Scenario:**

1. User selects the LBC scenario to execute from the EASE interview.
2. EASE schedules the execution of the LBC scenario with the correct input file provided by the user.
3. The user has the ability to view the execution of the model, if they choose to do so.
4. The output database is available for the user to conduct follow-on analysis.

## LBC-EASE Advanced User

**Primary Actor:** Study analyst in the ERS community (TRAC,MCCDC, etc.)
**Scope:** EASE, LBC

**Level:** User goal

**Stakeholders and Interests:**

- Study analyst— wants to execute LBC in EASE using a custom-built scenario
- EASE System Engineer— wants to integrate a LBC scenario within EASE based on the user's requirements

**Conditions:**

1. The modeler is able to create an LBC input file that matches the requirements of the scenario being considered. This means either having Microsoft Excel, Microsoft Access, or OpenOffice software available to create the database input file; alternatively, an appropriate XML editor if the input is chosen to be in XML format (not recommended).
2. The user has the ability to modify the scenario file.

**Success Guarantee:** The user's request is logged and scheduled, the LBC scenario executes without error, and the output files are available for analysis.
**Main Success Scenario:**

1. The modeler creates the LBC scenario (either MS Excel, MSAccess, Openofficedatabase, or XML format) to execute based on the requirements of the analysis being considered.
2. The modeler executes short test runs locally to ensure the correctness of the input database and correct any errors that are discovered.
3. EASE schedules the execution of the user-developed LBC scenario.
4. The modeler has the ability to view the execution of the model, if they choose to do so.
5. The output files from the model are available to conduct follow-on analysis.

# Appendix C
# COMBATXXI Run Model Script

## Batch Script Creation

The code below executes the process of reading in COMBATXXI model metadata and creating a batch script to execute in EASE.

```
import os
import lxml.etree
import shutil

#Current directory of the required files.
os.chdir("C:\Users\cmoten\Documents\Projects\CXXI-ERS\CXXIRequirements")

################################################################################
#                          XML Parsing and Logger Creation                     #
################################################################################

"""
This section of the code is to parse the xml files containing the study
parameters that can be adjusted. We will also parse the xml file for the
logger selections and  create the appropriate number of loggerdirectories
for each repetition of the scenario.
"""


#Read XML file for study parameters
#The parameters are STUDY, STOPTIME, REPS, and RANDSTREM
xmlTreeRoot = lxml.etree.parse('myXML.xml').getroot()
study = xmlTreeRoot.findtext('STUDY')
stopTime = int(xmlTreeRoot.findtext('STOPTIME'))
reps = int(xmlTreeRoot.findtext('REPS'))
randStream = xmlTreeRoot.findtext('RANDSTREAM')

logTree = lxml.etree.parse('logconfig.xml') #Logger Choices

#Change to the appropriate output directory
outputPath = 'C:\Users\cmoten\COMBATXXI\CXXIOutput'

#tree.write_c14n('output1.xml')

#Find the appropriate study
for cxxiRoot,cxxiDirectory,cxxiFile in os.walk('C:\Users\cmoten\COMBATXXI\Scenarios'):
    for files in cxxiFile:
        if files == study:
            studyPath = os.path.join(cxxiRoot,files)
```

```
#Make the ouput directory for the study log files
outputDirectory = os.path.join(outputPath,study)
if not os.path.exists(outputDirectory):
    os.makedirs(outputDirectory)
else:
 #removes all the currently existing subdirectories!
    shutil.rmtree(outputDirectory, ignore_errors=True)
    os.makedirs(outputDirectory)



#Make a directory for the logs for each repetition selected
for i in range(1,reps+1):
    folderName = 'Replication_%d_%s'%(i,randStream)
    folderPath = os.path.join(outputDirectory,folderName)
    fileName = 'tempLogConfig.xml'
    filePath = os.path.join(folderPath,fileName)
    os.makedirs(folderPath)
    logTree.write(filePath,encoding="UTF-8",xml_declaration=True)

################################################################################
#                              Batch File Creation                             #
################################################################################

"""
This section of the code writes the batch file to launch the appropriate
scenario. The stable build used for this script is 2014_08_14 and the jdk
for this build is jdk 1.8.0_40.
"""


#Template to write first few lines of batch file
template = """@echo off
cd C:\Users\cmoten\COMBATXXI\cxxi
set STUDY={STUDYPATH}
set STOPTIME={STOPTIME}
set RANDSTREAM={RANDSTREAM}
set path=%path%;C:\Users\cmoten\Java\jre1.8.0_40\\bin
set CXXIBASE=C:\Users\cmoten\COMBATXXI\cxxi
set CLASSPATH=%CXXIBASE%\\bin\*;%CXXIBASE%\lib\*
"""

"""
Set the studypath, stoptime, and randstream values from the xml file
information.
"""
context = {
 "STUDYPATH":studyPath,
 "STOPTIME":stopTime,
 "RANDSTREAM": randStream,
 }
```

```
classPath = "CLASSPATH"
stopTime = "STOPTIME"

with  open('out.bat','w') as myfile:
    myfile.write(template.format(**context))


#Code to make the replication text
for i in range(1,reps+1):
    temp1 = 'Replication_%d_%s'%(i,randStream)
    temp2 = os.path.join(outputDirectory,temp1)+'/tempLogConfig.xml'
    temp3 = "set LOG%d=" %i
    temp4 = temp3+temp2+"\n"
    temp5 = "set REP%d=%d\n" %(i,i)
    with open("out.bat", "a") as myfile:
        myfile.write(temp4)
        myfile.write(temp5)
for i in range(1,reps+1):
    temp6 = "java -cp %"+classPath+"%" +" -Xmx1G cxxi.model.CombatXXIModel -study %STUDY%"
    temp7 = " -rep %%REP%d%%" %i
    temp8 = " -stoptime %"+stopTime+"% -noview -standalone -CXXIBASE %CXXIBASE%"
    temp9 = " -logs %%LOG%d%%" %i
    temp10 = temp6 + temp7 +temp8 + temp9 + "\n"
    with open("out.bat","a") as myfile:
        myfile.write(temp10)
myfile.close()
```

# COMBATXXI Model Run Parameters

The parameters listed below are the minimum requirements to run a COMBATXXI model and were used as inputs in the previous section.

```
<parameters>
        <STUDY>KROE4_scenario.cxxi</STUDY>
        <STOPTIME>86400</STOPTIME>
        <REPS>3</REPS>
        <RANDSTREAM>SIMKIT_CONGRUENTIAL</RANDSTREAM>
</parameters>
```

**This page intentionally left blank.**

# Appendix D
# R Farmer Function

This appendix shows the current version of the farmer() function. It is hoped that the comments suffice to document it.

```
function (farmlist, remove.stuff = TRUE)
{
cat ("Start the farmer!\n")
start.time <- as.character (Sys.time ())
print (start.time)
timex <- substring (start.time, c(1, 6, 9, 12, 15), c(4, 7, 10, 13, 16))
timex <- paste (paste (timex[1:3], collapse=""), paste (timex[4:5], collapse=""), sep="-

require (XML)
#
# Set up defaults, then over-write them with the contents of
# GlobalVal. "Base" must have Windows-style backslashes, and
# may not end in a backslash. Also set global env. variables.
#
base      <- "h:\\trac\\combatxxi\\Model\\Scenarios\\KROE4_scenario"
cxxi.0    <- "KROE4_scenario.cxxi"
runner    <- "runner1.bat"
eval (parse (text = farmlist$GlobalVal))
Sys.setenv (RNG = "MERSENNE_TWISTER", STOPTIME = 86400,
LOGFILE = "h:\\trac\\CombatXXI\\Farming\\logconfig_1.xml",
OUTDIR = "h:\\trac\\CombatXXI\\Data\\TableDump")
#
# Step 1. Read in cxxi code.
#
cxxi.code <- scan (paste (base, "\\\\", cxxi.0, sep=""), sep="\n", what="", quiet=T)
scen.line <- grep ("SCENARIO=", cxxi.code)
scen.name <- strsplit (cxxi.code[scen.line], "=")[[1]][2]
#
# Here we rely on the scenario being in the same directory as the
# cxxi file.
#
origXML <- xmlParse (paste (base, "\\", scen.name, sep=""))
#
```

```
# Newscen will hold the scenario name; newcxxi will hold the full name
# of the manifest file, and newcxxi.base just the file name;/
# tempfiles will hold the name of the "deleteme" file.
#
newscen <- character (length (farmlist$Runs))
newcxxi <- newcxxi.base <- character (length (newscen))
tempfiles <- character (length (newscen))
#
# Loop one. Set up and write the scenario file (by modifying the XML) and the manifest
# by modifying the name of the scenario file). Of course we skip the first entry of the
# farmlist, which is the global stuff.
#
runs <- farmlist$Runs
for (i in 1:length (runs)) {
# For each run, set stuff as indicated in Settings
#
fxx <- origXML # make copy
objstrs <- paste ("//ScenarioObject[Name='", runs[[i]]$Settings$Name, "']/",
runs[[i]]$Settings$Property, sep="")
for (j in 1:length (objstrs)) {
thing <- xpathSApply (fxx, objstrs[j]) # pointer
xmlValue (thing[[1]]) <- runs[[i]]$Settings$Value[j]
}
#
# The "new" scenario and cxxi names have the form
# YYYYMMDD-HHMM-<runname>-<oldname>
#
newscen[i] <- paste (base, "\\", timex, "-", names(runs)[i], "-",
scen.name, sep="")
saveXML (fxx, newscen[i])
# Newscen has forward slashes, the way cxxi files do.
cat ("-- Saved XML to", newscen[i], "\n")
cxxi.code[scen.line] <-
paste ("SCENARIO=", newscen[i], "\n", sep="")
newcxxi.base[i] <-
paste (timex, "-", names (runs)[i], "-", cxxi.0, sep="")
newcxxi[i] <- paste (base, "\\", newcxxi.base[i], sep="")
write (cxxi.code, file = newcxxi[i])
cat ("-- Created new file: ", newcxxi[i], "\n")
tempfiles[i] <- tempfile (pattern = "ctemp", base,fileext = ".txt")
}
#
#
file.create (tempfiles)
#
```

```
# Now the fun part. Call "runner" once for each run. First set the
# global and local variables. (We need to reset globals in case they've
# been over-written by earlier locals.) Also set MANIFEST and DELETEME.
#
for (i in 1:length (runs)) {
if (length (farmlist$GlobalEnv) > 0)
do.call (Sys.setenv, farmlist$GlobalEnv)
if (length (runs[[i]]$LocalEnv) > 0)
do.call (Sys.setenv, farmlist$Runs[[i]]$LocalEnv)
Sys.setenv (MANIFEST=newcxxi[i], DELETEME=tempfiles[i])
cat ("Running with scenario", newscen[i], "\n")
cat ("...and delete file", tempfiles[i], "\n")
##    shell (paste (base, "\\", runner, sep=""),
##         shell="c:\\windows\\system32\\cmd.exe",
##         wait = FALSE)
cmd <- paste ("cmd.exe /c \"", paste (base, "\\", runner, sep=""), "\" >NUL 2>&1", sep="
system(cmd, intern = FALSE, ignore.stdout = TRUE,
ignore.stderr = TRUE,
wait = FALSE, input = NULL, show.output.on.console = FALSE,
minimized = TRUE, invisible = TRUE)
}
cat ("Jobs started!\n")
print (Sys.time())
start.loop.time <- Sys.time ()
#
# Now wait around until all the deleteme files are deleted.
#
while (1) {
elapsed <- as.numeric (Sys.time() - start.loop.time, "mins")
if (elapsed > 60) {
warning ("Abandoning waitloop after 60 minutes")
break
}
tempcount <- file.exists (tempfiles)
if (all (tempcount == FALSE))
{
cat ("Quitting wait loop successfully!\n")
break
}
cat ("Sleeping for 30!\n")
Sys.sleep (30)
}
cat ("Everything is done!\n")
if (remove.stuff == TRUE) {
file.remove (c(newcxxi, newscen))
```

```
}
#
# Now we have an output directory specified by OUTDIR which contains
# directories specified by the entries in newcxxi.base. Each of those
# directories has one or more directories with names like
# Replication_<n>_<rng_type>.
#
print (Sys.time())
return (list (outdir = Sys.getenv ("OUTDIR"),
dirs = newcxxi.base, farmlist = farmlist))
}
```

## Runner1.bat

The runner1.bat file (the "runner") is a DOS "batch" file that actually triggers a run of the simulation. It can be started from the Windows Explorer by double-clicking, or from the DOS command line. In this environment, the R farmer() command starts the runner by use of R's built-in system() command.

```
echo off

rem CLASSPATH and CXXIBASE are explicitly required, but we also
rem need to set INSTALL, which is used by the manifest itself.

if not defined CXXIBASE (set CXXIBASE=h:\trac\CombatXXI\Model\cxxi)
if not defined INSTALL  (set INSTALL=h:\trac\CombatXXI\Model\cxxi)
set CLASSPATH=%CXXIBASE%\bin\*;%CXXIBASE%\lib\*

rem Set other command-line arguments

if not defined RANDOMSTREAM (set RANDOMSTREAM=MERSENNE_TWISTER)
if not defined STOPTIME     (set STOPTIME=86400)
if not defined LOGFILE   (set LOGFILE=h:\trac\CombatXXI\Farming\logconfig_1.xml)
if not defined OUTDIR       (set OUTDIR=h:\trac\CombatXXI\Data\TableDump)

rem The MANIFEST variable holds the location of the cxxi file.

if not defined MANIFEST (set MANIFEST=h:\trac\CombatXXI\Model\ Scenarios\KROE4_scenario\

rem  Here's where we run. The caret is the continuation character.

java -cp %CLASSPATH% -Xmx1G cxxi.model.CombatXXIModel ^
CXXIBASE %CXXIBASE% -noview -standalone -stoptime %STOPTIME% ^
```

```
-randomstream %RANDOMSTREAM% -study %MANIFEST% ^
-logs %LOGFILE% -output %OUTDIR%

rem
if defined DELETEME (del %DELETEME%)
```

**This page intentionally left blank.**

# Appendix E
# References


[1] Secretary of Defense Robert M. Gates. *Science and Technology (S&T) Priorities for Fiscal Years 2013-17 Planning. Memorandum for Secretaries of the Military Departments*. Ed. by Washington D.C. Washington D.C., 2011. URL: `http://www.acq.osd.mil/chieftechnologist/publications/docs/OSD%2002073-11.pdf`.

[2] Eric Spero et al. "A Research Agenda for Tradespace Exploration and Analysis of Engineered Resilient Systems". In: *Procedia Computer Science* 28 (2014), pp. 763–772.

[3] Simon R Goerger, Azad M Madni, and Owen J Eslinger. "Engineered resilient systems: a dod perspective". In: *Procedia Computer Science* 28 (2014), pp. 865–872.

[4] Eric Spero et al. "Tradespace exploration for the engineering of resilient systems". In: *Procedia Computer Science* 28 (2014), pp. 591–600.

[5] Tommer Ender et al. *Enterprise Architecture Tradespace Analysis*. Tech. rep. ADA603143, XD, DOD, SERC-2014-TR-043, H98230-08-D-0171. Georgia Instittute of Technology, Atlanta, GA, Feb. 2014, p. 21.

[6] Jefferey P. Holland. *Engineered Resilent Systems (ERS)*. Presentation to Ms. Blechinger. 2014.

[7] TRADOC Analysis Center-White Sands Missile Range. *COMBATXXI Users Guide*. 2014. URL: `https://cxxi.wsmr.army.mil/Welcome.html`.

[8] TRADOC Analysis Center-Fort Lee. *DM/LBC Users Guide v.3.15.0*. 2012.

[9] Cockburn Alistair. *Writing effective use cases*. Boston, MA: Addison-Wesley, 2001.

[10] Chris Gaughan et al. "Executable Architecture Systems Engineering". In: *80th MORS Symposium, Working Group 29*. 2013.

[11] Shaun Murphy et al. "U.S. Army Modeling and Simulation Executable Architecture Deployment Cloud Virtualization Strategy". In: *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*. IEEE, 2012, pp. 880–885.

[12] Shaun Murphy, Manuel Diego, and Scott Gallant. "U.S. Army Modeling and Simulation Executable Architecture Deployment Cloud Using Virtualization Technologies to Provide and Extensible Platform as a Service (PaaS) Cloud for Federated Application Configuration and Execution". In: *Fall Simulation Interoperability Workshop, 11F-SIW-051*. 2011.

[13] Chris Gaughan et al. "When Tradespace Analysis Met Combat Modeling and Simulation". In: *I/ITSEC, Session Number: 14264*. 2014.

[14] Chris Gaughan et al. "Bringing Next Generation Simulation into the Land of Practicality". In: *The Interservice/Industry Training, Simulation & Education Conference (I/ITSEC)*. Vol. 2013. NTSA, 2013. Chap. 1.

[15] TRADOC Analysis Center-White Sands Missile Range. *COMBATXXI Wiki*. 2014. URL: `https://cxxi.wsmr.army.mil/Welcome.html`.

[16] Julia Piotto and Richard Dexter. *Defining the Structure of the COMBATXXI Databases*. Tech. rep. ADB392433, DODA, X5, AR-015-600, DSTO/LOD, DSTO-GD-0744, DEFENCE SCIENCE and TECHNOLOGY ORGANISATION EDINBURGH (AUSTRALIA) LAND OPERATIONS DIV, Apr. 2013, p. 349.

[17] Peter A Nesbitt et al. *Knowledge Representation for Decision Making Agents*. Tech. rep. ADA589932, XA, TRAC-WSMR*, TRAC-M-TR-13-054, ARMY TRADOC ANALYSIS CENTER MONTEREY CA, July 2013, p. 88.

[18] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2015. URL: `https://www.R-project.org/`.

[19] Python Software Foundation. *Python Language Reference, version 2.7*. 2015. URL: `http://www.python.org`.

[20] Gary Horne and Ted Meyer. "Data farming and defense applications". In: (2010).

# Appendix F
# Glossary

| | |
|---|---|
| Actor | The user or developer of the combat model. |
| AMSAA | Army Material Systems Analysis Activity |
| ARL | Army Research Laboratory |
| COMBATXXI | Combined Arms Analysis Tool for the 21st Century |
| Conditions | The circumstances related to pre and post execution of the scenario. |
| COI | Community of Interest |
| Customizable parameters | The options within a scenario file the user can modify for a specific purpose. |
| DoD | Department of Defense |
| EASE | Executable Architecture Systems Engineering |
| ERDC | Engineer Research Development Center |
| ERS | Engineered Resilient Systems |
| Extensions | Potential contingencies for the scenario integrator or EASE system engineer to manage. |
| FACT | Framework for Assessing Cost and Technology |
| GTRI | Georgia Tech Research Institute |
| GUI | Graphical User Interface |
| JCIDS | Joint Capabilities Integration Development System |
| JDAFS | Joint Dynamic Allocation of Fires and Sensors |
| JDK | Java Development Kit |
| JRE | Java Runtime Environment |
| JVM | Java Virtual Machine |
| LBC | Logistics Battle Command |
| Main success scenario | What would happen if the scenario executed with no errors. |
| MCCDC | Marine Corps Combat Development Command |
| MSBL | Maneuver Support Battle Laboratory |
| OPMODE | Operational Mode |
| OPTEMPO | Operations Tempo |
| OneSAF | One Semi-Automated Forces |
| Operating requirements | The elements that are critical to the proper execution of a combat model. |
| PaaS | Platform as a Service |
| Scope | The system(s) related to the use case. |
| SFF | Standard File Format |
| SITS | Scenario Integration Tool Suite |

| | |
|---|---|
| Stakholder | Someone who has a vested interest in the proper execution of the combat model in EASE. |
| TRAC | Training and Doctrine Command Analysis Center |
| Use Case | The document describing how a user will interact with EASE and a combat model to accomplish a specified goal. |
| TRAC-WSMR | TRAC-White Sands Missile Range |
| XML | Extensible Markup Language |