ADVANCED FRAMEWORK FOR SIMULATION, INTEGRATION
AND MODELING (AFSIM) Version 1.8 OVERVIEW

James Zeh, AFRL/RQQD
Brian Birkmire, AFRL/RQQD

Peter D. Clive
Jeffrey Johnson
Andrew W. Krisby
Jonathon E. Marjamaa
Luke B. Miklos
Michael J. Moss
Stephen P. Yallaly

The Boeing Company

Oct 2014
TECHNICAL Report
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of Figures</td>
<td>ii</td>
</tr>
<tr>
<td>List of Tables</td>
<td>iii</td>
</tr>
<tr>
<td>1.0 INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Background</td>
<td>1</td>
</tr>
<tr>
<td>2.0 AFSIM SOFTWARE SUITE</td>
<td>2</td>
</tr>
<tr>
<td>2.1 Functional Architecture</td>
<td>2</td>
</tr>
<tr>
<td>2.2 VESPA</td>
<td>4</td>
</tr>
<tr>
<td>2.3 AFSIM Integrated Development Environment (IDE)</td>
<td>5</td>
</tr>
<tr>
<td>2.3.1 General Plug-in Capability</td>
<td>6</td>
</tr>
<tr>
<td>2.3.2 Experiment Parameter Comment Tag Plug-in</td>
<td>7</td>
</tr>
<tr>
<td>2.3.3 Code Folding</td>
<td>8</td>
</tr>
<tr>
<td>2.3.4 The Command Chain Browser</td>
<td>9</td>
</tr>
<tr>
<td>2.3.5 Regression Testing plug-in</td>
<td>11</td>
</tr>
<tr>
<td>2.4 RIRR</td>
<td>13</td>
</tr>
<tr>
<td>2.4.1 OVERVIEW</td>
<td>13</td>
</tr>
<tr>
<td>2.4.2 COGNITIVE MODEL</td>
<td>14</td>
</tr>
<tr>
<td>2.4.3 QUANTUM TASKER</td>
<td>14</td>
</tr>
<tr>
<td>2.4.4 BEHAVIOR TREE</td>
<td>15</td>
</tr>
<tr>
<td>2.4.5 ROUTE FINDER</td>
<td>16</td>
</tr>
<tr>
<td>2.4.6 CLUSTER MANAGER</td>
<td>16</td>
</tr>
<tr>
<td>2.4.7 EXAMPLE AGENT INTERACTION USING ALL TECHNOLOGIES</td>
<td>17</td>
</tr>
<tr>
<td>3.0 AFSIM IADS MODEL</td>
<td>17</td>
</tr>
<tr>
<td>4.0 SUMMARY</td>
<td>18</td>
</tr>
<tr>
<td>Appendix A - AFSIM Concept of Operations</td>
<td>19</td>
</tr>
<tr>
<td>Appendix B - AFSIM Communication, Sensor and Jamming Equations</td>
<td>34</td>
</tr>
<tr>
<td>Appendix C - AFSIM Synthetic Radar Equations</td>
<td>50</td>
</tr>
<tr>
<td>Appendix D - AFSIM Electronic Warfare Overview</td>
<td>57</td>
</tr>
<tr>
<td>Acronyms</td>
<td>68</td>
</tr>
</tbody>
</table>
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1.</td>
<td>AFSIM Functional Architecture</td>
<td>2</td>
</tr>
<tr>
<td>Figure 2.</td>
<td>The VESPA GUI</td>
<td>4</td>
</tr>
<tr>
<td>Figure 3.</td>
<td>Current AFSIM Constructive Analysis Process</td>
<td>5</td>
</tr>
<tr>
<td>Figure 4.</td>
<td>Future AFSIM Constructive Analysis Process</td>
<td>6</td>
</tr>
<tr>
<td>Figure 5.</td>
<td>The Plug-in Manager User-Interface</td>
<td>7</td>
</tr>
<tr>
<td>Figure 6.</td>
<td>Experiment Parameter Comment Tag Plug-in</td>
<td>8</td>
</tr>
<tr>
<td>Figure 7.</td>
<td>Code Folding Feature</td>
<td>8</td>
</tr>
<tr>
<td>Figure 8.</td>
<td>Collapse and Expand Icons</td>
<td>9</td>
</tr>
<tr>
<td>Figure 9.</td>
<td>Command Chain Browser to Manipulate Platform Relationships</td>
<td>9</td>
</tr>
<tr>
<td>Figure 10.</td>
<td>Infinity Icon Cyclical Relationship Indication</td>
<td>10</td>
</tr>
<tr>
<td>Figure 11.</td>
<td>The Object Mode Banner</td>
<td>10</td>
</tr>
<tr>
<td>Figure 12.</td>
<td>Regression Testing Input Dialog</td>
<td>11</td>
</tr>
<tr>
<td>Figure 13.</td>
<td>Regression Test HTML Report</td>
<td>13</td>
</tr>
<tr>
<td>Figure 14.</td>
<td>Quantum Tasker Method of Operation</td>
<td>15</td>
</tr>
<tr>
<td>Figure 15.</td>
<td>RIPR Behavior Tree</td>
<td>15</td>
</tr>
<tr>
<td>Figure 16.</td>
<td>RIPR Route Finder</td>
<td>16</td>
</tr>
<tr>
<td>Figure D-1.</td>
<td>EW Techniques Architecture</td>
<td>57</td>
</tr>
<tr>
<td>Figure D-2.</td>
<td>Jamming system architecture in AFSIM showing EA techniques</td>
<td>58</td>
</tr>
<tr>
<td>Figure D-3.</td>
<td>Radar System Architecture in AFSIM Showing EP techniques</td>
<td>59</td>
</tr>
<tr>
<td>Figure D-4.</td>
<td>Mapping of EA &amp; EP Interactions in the AFSIM EW Architecture</td>
<td>59</td>
</tr>
<tr>
<td>Figure D-5.</td>
<td>Hierarchy of Base Type Effects for EA</td>
<td>61</td>
</tr>
<tr>
<td>Figure D-6.</td>
<td>Hierarchy of Base Type Effects for EP</td>
<td>62</td>
</tr>
<tr>
<td>Figure D-7.</td>
<td>AFSIM EW Interaction Flowchart</td>
<td>67</td>
</tr>
</tbody>
</table>
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table B-1. Transmitted Power Variables</td>
<td>35</td>
</tr>
<tr>
<td>Table B-2. Free Space Propagation Variables</td>
<td>36</td>
</tr>
<tr>
<td>Table B-3. Free Space Reflected Signal Variables</td>
<td>37</td>
</tr>
<tr>
<td>Table B-4. Free Space Received Power Variables</td>
<td>37</td>
</tr>
<tr>
<td>Table B-5. Bandwidth Ratio Variables</td>
<td>38</td>
</tr>
<tr>
<td>Table B-6. Receiver Power Variables</td>
<td>38</td>
</tr>
<tr>
<td>Table B-7. Antenna Gain Variables</td>
<td>40</td>
</tr>
<tr>
<td>Table B-8. Processed Power Variables</td>
<td>42</td>
</tr>
<tr>
<td>Table B-9. Signal and Noise Variables</td>
<td>43</td>
</tr>
<tr>
<td>Table B-10. Passive Receiver Noise Variables</td>
<td>44</td>
</tr>
<tr>
<td>Table B-11. SAR Collection Time Variables</td>
<td>45</td>
</tr>
<tr>
<td>Table B-12. Communications Signal and Noise Variables</td>
<td>47</td>
</tr>
<tr>
<td>Table B-13. IRST Contrast Radiant Intensity Variables</td>
<td>48</td>
</tr>
<tr>
<td>Table B-14. IRST Probability of Detection Variables</td>
<td>49</td>
</tr>
<tr>
<td>Table C-1. Radar Range Equation Variables</td>
<td>50</td>
</tr>
<tr>
<td>Table C-2. Receiver Noise Variables</td>
<td>51</td>
</tr>
<tr>
<td>Table C-3. SAR Dwell Time Variables</td>
<td>52</td>
</tr>
<tr>
<td>Table C-4. Range Processing Gain Variables</td>
<td>53</td>
</tr>
<tr>
<td>Table C-5. Resolution Cell Variables</td>
<td>54</td>
</tr>
<tr>
<td>Table C-6. Pseudo-Image Generation Variables</td>
<td>55</td>
</tr>
<tr>
<td>Table D-1. Effect Coherency Types</td>
<td>63</td>
</tr>
<tr>
<td>Table D-2. Jamming Power Types</td>
<td>63</td>
</tr>
<tr>
<td>Table D-3. Jamming Effects Variable Structure</td>
<td>64</td>
</tr>
<tr>
<td>Table D-4. Signal Effects Variable Structure</td>
<td>65</td>
</tr>
<tr>
<td>Table D-5. Track Effects Variable Structure</td>
<td>65</td>
</tr>
<tr>
<td>Table D-6. Message Effects Variable Structure</td>
<td>66</td>
</tr>
<tr>
<td>Table D-7. Aggregation Types</td>
<td>66</td>
</tr>
</tbody>
</table>
1.0 INTRODUCTION

AFSIM is a government-approved C++ simulation framework for use in constructing engagement and mission-level analytic simulations for the Operations Analysis community, as well as virtual experimentation. The primary goal of AFSIM applications is the assessment of new system concepts and designs with advanced capabilities not easily assessed within traditional engagement and mission level simulations. Development activities include modeling weapon kinematics, sensor systems, electronic warfare systems, communications networks, advanced tracking, correlation, and fusion algorithms, and automated tactics and battle management software.

In this section, the reasons for the development and history of AFSIM are presented. Section 2 provides an overview of AFSIM architecture and software suite. In Section 3, the AFSIM IADS model is described followed by a discussion of the challenges associated with performing mission-level constructive analysis with large scenarios, such as those containing large, complex IADS models. Section 4.0 provides a quick summary. The four appendixes describe AFSIM concept of operation and model details.

1.1 Background

AFSIM is based on The Boeing Company’s Analytic Framework for Network-Enabled Systems (AFNES). Under contract, Boeing delivered AFNES to the Air Force (specifically AFRL/RQQD) with unlimited government rights, including source code, in Feb 2013. AFRL/RQQD rebranded AFNES as AFSIM and has started to distribute AFSIM within the Air Force and DoD, including DoD contractors.

The Boeing Company developed and funded the AFNES simulation framework through internal research and development (IR&D) funding from 2003-2014. Beginning in 2005, Boeing began developing a customized AFNES capability to simulate threat Integrated Air Defense Systems (IADS) to assess advanced air vehicle concepts performing Precision Engagement missions. The requirements of this new IADS simulation capability included being able to match results with the Air Force-approved mission level model. The reason for developing an AFNES alternative to the Air Force IADS modeling capability relates to the limitations associated with the Air Force mission level model. Examples of areas in which the Air Force mission level model is lacking include: expansion of representations of Electronic Warfare (EW) techniques; the integration of independent tracking and correlation systems; utilization of vendor-supplied auto-routers and mission optimization capabilities; net-centric communications systems; the contribution of Space assets; and integration of special, existing models, such as AGI’s System Tool Kit (STK).

The AFNES IADS capability became operational in 2008, and is currently being utilized by multiple Boeing development programs, as well as government contracted programs, to assess the ability of advanced air vehicle design concepts to penetrate advanced Air Defense networks and conduct precision engagement missions. Because the Air Force is also interested in analyzing future vehicles in the area of persistent and responsive precision engagement, this capability has been briefed to Air Force personnel at various times over the previous five years. In 2010, the
AFRL/RQQD Aerospace Vehicles Technology Assessment & Simulation (AVTAS) Lab (formerly AFRL/RBCD) commissioned a trade study of M&S Frameworks for the purpose of assessing potential alternatives to replace or augment their current constructive simulation environment. The result of the AFRL trade study was the selection of AFNES as the best M&S framework to meet their air vehicle mission effectiveness analysis requirements.

2.0 AFSIM SOFTWARE SUITE

2.1 Functional Architecture

AFSIM is an OO, C++ simulation environment that facilitates the rapid prototyping of customized engagement and mission-level warfare simulations. AFSIM includes a set of software libraries, shown as a functional architecture in Figure 1, containing routines commonly used to create analytic applications. The AFSIM infrastructure includes routines for the top-level control and management of the simulation; management of time and events within the simulation; management of terrain databases; general purpose math and coordinate transformation utilities; and support of standard simulation interfaces, such as those supporting the Distributed Interactive Simulation (DIS) protocol. The AFSIM component software routines support the definition of entities (platforms) to populate scenarios. These software routines contain models for a variety of user-defined movers, sensors, weapons, processors for defining system behavior and information flow, communications and track management. Appendix A, AFSIM Concept of Operations, provides a more comprehensive description of the AFSIM architecture, its applicability, and how users interact with applications that utilize the framework.

![AFSIM Functional Architecture](image)

The top-level characteristics and capabilities of the AFSIM framework include:
- A class hierarchy of simulation objects, including data-driven platforms, movers, sensors, communications networks, processors, weapons, and simulation observers.
• Simulation class and Event class controlling the time or event step operation of AFSIM models, and the logging of entities within the simulation.
• Standard math libraries for coordinate systems (WGS-84, Spherical, ENU, NED), random number generation, DIS communication, High-Level Architecture (HLA) publish and subscribe, and generalized software routines, such as container classes for storing objects and data.
• A common geo-spatial environment and terrain representation, importing standard formats such as National Geospatial-Intelligence Agency (NGA) Digital Terrain Elevation Data (DTED), ESRI, GeoTiff and VMAP database formats.
• A general-purpose scripting language developed to allow limited access to the framework using text input files (i.e., scripts) rather than through the Application Programming Interface (API).
• Communications network modeling, including basic radio transceivers and advanced communications algorithms, including addressable nodes, routers, multi-access protocols, contention and queuing.
• Electronic warfare modeling, including noise and deceptive jamming techniques, as well as the ability to jam and degrade any type of electro-magnetic receiver, including communications systems.
• Modeling of information flow and tasking between player and system elements to define candidate NCO concepts.
• The ability to run any AFSIM application in both constructive (batch processing) and virtual (real-time) modes.
• User interface elements for integrated scenario generation and post-processor visualization software.

Appendix B provides details of the equations used within AFSIM to model communications, sensors and jamming. Appendix C provides additional details of the equations used within AFSIM to model the synthetic aperture radar (SAR). Appendix D provides additional details of the AFSIM Electronic Warfare architecture.

In addition to the AFSIM core, several capabilities are available. Additional capabilities include: multitarget tracking algorithms; Link-16 modeling of both the physical and message layers; and Reactive Integrated Planning aRchitecture (RIPR) intelligent agent algorithms for implementing complex object behaviors. RIPR utilizes a Boeing-developed Quantum Tasker concept for commander subordinate interaction and task de-confliction. Section 3 provides additional details of the RIPR model. Restricted capabilities include missile flyout models.

The baseline AFSIM constructive application is called the Simulation of Autonomously Generated Entities (SAGE), which was one of the first constructive applications developed using the AFSIM framework. SAGE is a simple application that reads in a user-defined input file, executes the simulation, and outputs any user-defined data files. The original purpose for SAGE was to simulate background air, road or maritime traffic. Although SAGE retains the capability to generate background traffic, the user can exercise all of the resident AFSIM capabilities. The entire AFSIM IADS model is executed using the SAGE application.

Distribution A: Approved For Public Release (Case #:88ABW-2014-5041)
2.2 VESPA

To support the analyst, Boeing developed tools to facilitate scenario generation and post-process data analysis and visualization. Specifically, the Visual Environment for Scenario Preparation and Analysis (VESPA) software application was developed to support the creation of scenario initial condition files compatible with any AFSIM-based application. In addition, VESPA can be used to visualize object positional time histories and other event information generated as output from any AFSIM-based application. This allows the analyst to quickly understand and analyze the output from the simulation. Since VESPA is a “DIS-listener” visualization tool, it may also be used to display real-time entity interactions from any real-time simulation that publishes DIS data.

VESPA includes a graphical user interface (GUI) that includes a drawing area with a geospatial map and a data input area, as shown in Figure 2.

Figure 2. The VESPA GUI

Using VESPA, the analyst can place icons representing objects at specific latitude and longitude locations on a geospatial map. Initial conditions can then be assigned for each selected object. For example, the initial conditions of an aircraft could be its speed, heading and altitude. Visual features associated with objects, called attachments, can also be created. Examples include routes, range rings and zones.

VESPA can be used to display object positional histories and events using an AFSIM replay file generated during an AFSIM simulation run. The AFSIM replay file is a binary file containing the DIS output from the AFSIM simulation. In addition, plots can be generated for selected events that occurred during the simulation.
The process of using VESPA to create the scenario initial conditions for input into an AFSIM based application and visualizing post-process results is illustrated in Figure 3.

![Figure 3. Current AFSIM Constructive Analysis Process](image)

2.3 AFSIM IDE

AFSIM permits the user to create subsystem definitions in separate files and to simply include the filename as part of the buildup of a more complex system definition. This enables subsystem configuration control and reuse. This flexibility leads to large numbers of subsystem definition files when creating scenarios with a wide variety of different complex systems. The VESPA application facilitates the creation of the scenario initial conditions files. It does not, however, address the problems associated with defining and integrating system and subsystem models or defining system-level relationships such as command chains and peers using ASCII data files. Any input file errors are not found until an AFSIM application is executed. In some cases, the AFSIM error messages may not adequately indicate the location of input file errors for the analyst.

In early 2011, Boeing initiated the development of the AFSIM Integrated Development Environment (IDE) to support the analyst in defining and integrating system and subsystem models. The AFSIM IDE patterns itself on IDEs created for use with software development. With software IDEs, a single application is used to edit files, compile, link, and run the software executable, and view output results or error messages. Likewise, the AFSIM IDE permits the analyst to edit input files, execute the AFSIM application, and visualize the output results and any error messages. The iterative process that allows the analyst to receive immediate feedback as system and subsystem models are defined and scenarios created is illustrated in Figure 4.
Current capabilities of the AFSIM IDE to support input file creation include file syntax highlighting, auto-completion, context-sensitive command documentation and a variety of scenario browsers. Syntax highlighting makes reading and understanding the content easier for the analyst. Unknown keywords or commands are underlined in red for easy discovery. Examples of unknown keywords or commands include misspelling of keywords or using keywords out of scope. The auto-completion feature provides a list of suggestions for the analyst to choose from, based on the context. The analyst can select one of the suggestions, and the command will be completed without having to manually type the command. Context-sensitive command documentation allows the analyst to bring up documentation associated with a command to illustrate the scope and use of the command. Other AFSIM IDE capabilities are available to assist the analyst in defining system and subsystem models and scenarios.

The AFSIM IDE can execute any AFSIM-based application using the input files defined by the analyst. Any screen output from the application is displayed in an IDE output window along with any error messages.

Current capabilities of the AFSIM IDE to view AFSIM simulation results include the ability to run the VESPA application from the IDE using the AFSIM replay file created during the simulation run.

2.3.1 General Plug-in Capability

A general plug-in capability is part of the AFSIM IDE. This supports the development of features that are needed by a limited user set. This feature includes the plug-in interface to the application, and user-interface to manage plug-ins within the applications.
Plug-ins are kept in a “plug-ins” subdirectory of the IDE installation directory. Plug-ins will work on both Windows and Linux operating systems. On Windows, the plug-ins are dynamic link libraries (DLL), and on Linux they are shared objects (SO).

In the AFSIM IDE, the plug-in manager user-interface, as shown in Figure 5, can be accessed from the ‘Options’ menu by selecting ‘Plug-in Manager’.

![Figure 5. The Plug-in Manager User-Interface](image)

The plug-in manager contains a list of available plug-ins in a list on the left. To the right of the list there is a text description of the plug-in that is currently selected. Below the description is a check-box labeled Auto-start. New plug-ins put in the plug-ins folder will auto-load by default. When checked, the currently selected plug-in will automatically be loaded in future sessions of the IDE. The check-box does not affect the current state of the plug-in. The button on the right will allow the user to either load or unload the plug-in, depending on the current state. The OK button on the very bottom of the manager is used to close the plug-in manager.

Plug-ins have access to the entire application, but developers should be encouraged to put the initial user-interface in the Plug-ins menu. This menu will display only when a plug-in requests it.

### 2.3.2 Experiment Parameter Comment Tag Plug-in

The experiment parameter plug-in allows users to contain selections of text inside of comment tags. The comment tags will not affect a SAGE simulation, but can be used by a pre-
processor, such as PIANO, to modify the scenario in support of design of experiments (DOE) applications.

To use the plug-in, a developer would first highlight the text of interest, then right-click to display the context-sensitive menu. Selecting *Add Experiment Parameter* will open the Create Experiment Parameter dialog box as shown in Figure 6. This box contains a text field in which a tag may be defined. The *OK* button will complete the operation, and the *Cancel* button will terminate it. Once the operation is complete, the selected text will be framed by the comment tag. The tag will open with the text “/*{{" and close with “/*TAG}}*/”, where “TAG” is the tag value defined in the Create Experiment Parameter dialog box.

### 2.3.3 Code Folding

This feature allows the developer to show and hide blocks of text in their display as shown in Figure 7.
The IDE automatically recognizes blocks of text representing structural elements of the AFSIM input language. In the left margin of the text-editor, these blocks will be decorated with a collapse icon, as shown in Figure 8, which, when clicked with the left mouse button, will collapse the block of text.

![Collapse and Expand Icons](image)

**Figure 8. Collapse and Expand Icons**

Collapsed blocks of text will maintain the first line of the block decorated by an expand icon which, when clicked with the left mouse button, will return the block of text to its original state.

2.3.4 The Command Chain Browser

The command chain browser, as shown in Figure 9, is an IDE panel that allows developers to quickly understand and manipulate the command relationships of platforms in their scenarios.

![Command Chain Browser](image)

**Figure 9. Command Chain Browser to Manipulate Platform Relationships**

The command-chain browser, by default, will load when the application starts along with the platform and type browser on the left. The browser may be loaded, or brought to the front of the screen, by clicking on the View menu and selecting Command-Chain Browser.
At the top of the command-chain browser is a drop-down menu that controls the side filter. Selecting a team from this menu will limit the displayed relationships to that team. Below this is the chain filter drop-down menu. Selecting a command chain from this menu will limit the displayed relationship to those in that command chain. The \( \times \) button to the right of the side-filter menu will return the browser to a completely unfiltered state.

Below the filters is the command-chain tree. Objects “low” in the tree represent commanders, and objects attached to the commanders are subordinates. Platforms are color-coded by team. This color will match the team name; the team name is a color in all lower case, and will be randomly assigned in other cases. Platforms are assigned an icon that should represent their domain as determined by their mover.

Command chains are at the lowest level of the tree. Platforms that are unassigned (to SELF or another platform) in a given command-chain will be put in the Unassigned Platforms branch. This branch is labeled with the number of players contained within in parenthesis.

If the IDE detects a cyclical relationship it will assign the platform a special icon, as shown in Figure 10, and will only show the subordinate relationships until it finds itself.

![Figure 10. Infinity Icon Cyclical Relationship Indication](image)

Command relationship may be changed by dragging a player from the command chain onto another player on the command chain. When this is done, text editing will be locked and a banner will be displayed as shown in Figure 11. Changes must be applied or discarded before text editing may continue.

![Figure 11. The Object Mode Banner](image)

2.3.5 Regression Testing plug-in

The regression testing plug-in allows developers to compare the results of AFSIM runs with baseline results. The capability consists of three pieces: a user interface used to define the test, a python script that executes the test, and a generated HTML report.
It is not required to use the input dialog to perform regression tests; the test script is executable from the command line.

The regression testing input dialog, as shown in Figure 12, allows the user to define a batch of tests. At the top of the dialog is a text input used to define the batch name, along with buttons to open an existing batch, save the batch, save the batch with a particular name, or execute the batch. Developers may define the number of processors to use in execution. If any executions are dependent on prior executions, the developer should limit themselves to one processor.

The filtering, comparisons and report generation are performed by a python script, so a version of python is required. The *Python Exe* text input and browse buttons allow the developer to define this. The script was developed with Python 3.2.

The script generates a number of files, including filtered inputs and the report, which will be written to an output directory as defined by the *Output Directory* text input and browser buttons.

In the *Executions* group, the user may define the executions needed to generate test data. The buttons to the left allow the developer to add new executions, remove existing executions,
and reorder the executions. The check-box on the individual executions allows them to be enabled or disabled in the batch. Executions have names, executables, working directories and parameters.

Outputs may be filtered before the comparisons are performed. The *Filter Expressions* group contains these filters. The buttons to the left of the group allow the developer to add new filters, remove existing filters, or reorder the filters. Filters have a name, for identification, and a regular expression.

A test is a set of comparisons. The *Tests* group allows the developer to manage the tests they want to run for their test batch. Tests may be added, removed, or reordered. The check box on a given test determines whether it will be run in the batch. Tests have names. The selected test will affect what is displayed in the *Comparisons* group.

A comparison is an individual file comparison. The *Comparison* group allows the developer to manage these by adding new comparison to the test, removing existing comparison, and reordering comparisons. Individual comparison may be enabled or disabled for the selected test using the check-box. A comparison includes a baseline filename, and an output file.

The *Filters* group determines which filters will be applied to a given comparison. Filters may be enabled or disabled by their check-boxes.

The *Messages* group will display output relating to the success of the batch execution.

After execution, an HTML report will be generated and displayed.
The report will display the results for the batch. At the top of the report is a summary. Below the summary are the results for the individual comparisons. The comparisons include hyperlinks to display the filtered and unfiltered baseline and output files. If the comparison failed, the developer may click Show Differences... to display the file with differences highlighted in red. Differences that begin with ‘-‘ indicate that a line was removed from the baseline, and those beginning with ‘+’ indicate an addition from the baseline. An example regression test HTML report is illustrated in Figure 13.

2.4 Reactive Integrated Planning aRchitecture (RIPR)

2.4.1 Overview

RIPR is the framework included with AFSIM that enables behavior modeling. RIPR is agent based, meaning that each agent acts according to its own knowledge; however, it is common for agents to cooperate and communicate with each other. RIPR is best thought of as a collection of utilities and algorithms that tie together nicely in the construction of an intelligent agent. Most modern RIPR agents, however, do contain a Perception Processor and a Quantum Tasker Processor. The agent senses the world by querying the platform and its subsystems, for information. The agent builds knowledge internally, makes decisions, and then takes action by controlling its platform accordingly. Most platform queries and control actions take place inside...
of the AFSIM scripting language. The knowledge-building and decision-making actions that RIPR performs are aided by various artificial intelligence technologies summarized here.

2.4.2 Cognitive Model

A RIPR agent maintains its own perception of threats, assets, and peers. This represents an agent’s limited brain and the information can be delayed or erroneous. To represent players of varying skill, each agent has its own tunable cognitive model. For example, an “expert” pilot agent can maintain knowledge of 16 threats that he updates (looks at radar) every 5 seconds. Much of the cognitive model’s ability is contained within the Perception Processor.

2.4.3 Quantum Tasker

The RIPR Quantum Tasker is used for commander subordinate interaction and task de-confliction. The Quantum Tasker comprises task generator(s), task-asset pair evaluator(s), an allocation algorithm, and various strategy settings (such as how to handle rejected task assignments). Each component (generator, evaluator, allocator) can be selected from pre-defined options, or custom created in script. The RIPR Quantum Tasker tasking system is also compatible with platforms using the older task manager (WSF_TASK_MANAGER and WSF_TASK_PROCESSOR). It can send and/or receive tasks to/from other RIPR agents and other task manager platforms.

The Quantum Tasker’s method of operation:

- Acquire perception of assets from cognitive model for matrix columns
- Acquire perception of threats from cognitive model
- Generator generates tasks for matrix rows
- Strategy dictates how previously assigned tasks, rejected tasks, or new tasks are handled.
- Evaluator calculates values for possible asset-task pairs for matrix body
- The allocator runs on the task-asset matrix to find appropriate task allocation, e.g. greedy, optimal, etc
- Tasks are assigned over comm, handshaking performed for acceptance/rejection
2.4.4 Behavior Tree

RIPR agents typically make use of a RIPR behavior tree to define their behavior. A behavior is a compact modular piece of script that performs some unique action. Behaviors should be parameterized and reusable. A behavior tree allows connection of behaviors in interesting ways so they perform in certain orders or subsets. The whole tree aggregates the behaviors to model an agent’s behavior.
RIPR behavior trees provide five different intermediate connector-node types:

- **Selector** – chooses and performs first child behavior to pass its precondition check.
- **Sequence** – performs all child behaviors in sequence until one fails its precondition check.
- **Parallel** – performs all child behaviors whose precondition check passes.
- **Weight Random** – makes a weighted random selection from its child behaviors. Weight based on precondition value returned.
- **Priority Selector** – selects the child behavior who returns the largest precondition value.

Behavior trees provide for maximum utility in developing and editing agents. A properly constructed behavior tree allows a user to find relevant script fast, and swap in other behaviors at appropriate places. For example: try separating out behaviors for choosing desired heading, altitude, and speed from the behavior that actually performs the flight task. When you develop a new flying behavior, e.g. one that used a new route finder, you can swap that for the old one while keeping the logic in place for calculating desired direction.

### 2.4.5 Route Finder

The route finder allows an agent to path around static and/or dynamic obstacles. At this time, obstacles are circular regions defined at a location or attached to another platform. The route finder takes advantage of this assumption to quickly build a search graph around all avoidances and uses a depth-first-search to find the best routes to a target. The route finder has three options for impossible routes:

- Shrink offending avoidances
- Ignore offending avoidances
- Shift start/target points outside of all offending avoidances.

![Figure 16. RIPR Route Finder](image)

### 2.4.6 Cluster Manager

Some RIPR agents take advantage of the Cluster Manager to perform clustering on threat or asset perception in order to think of these larger sets as smaller groups. For example, it is
common for a commander to group incoming threats into two clusters so it can send each of its
two squadrons after separate groups. The Cluster Manager can cluster based on desired similarity
thresholds or based on the desired number of clusters. Similarity measurements can be based on
ground distance, 3D distance, or 3D distance and speed. The Cluster Manager can use one of
three clustering algorithms:

- Hierarchical Tree Max – default, guaranteed to be optimal, no cluster member
dissimilar to any other member past the threshold (this method provides for tighter
“classic” groups of members)
- Hierarchical Tree Min – guaranteed to be optimal, no cluster member dissimilar to at
least one other member past the threshold (this method allows for long “stringy”
chains of members)
- K-Means – not guaranteed to be optimal, fastest, clusters are centered on K different
mean points.

2.4.7 Example agent interaction using all technologies

- A commander agent obtains threats from his cognitive model (Perception Processor)
- Commander’s Quantum Tasker generator clusters threats into groups and creates a
task for each group
- Commander’s Quantum Tasker evaluator scores his squadrons (assets) against each
group
- Commander’s Quantum Tasker allocator finds optimal task assignment
- Commander assigns task(s) to subordinate flight leads over comm.
- Flight lead uses asset and threat perception from cognitive model while interpreting
task
- Flight lead agent’s Quantum Tasker generates, evaluates, allocates, and assigns tasks
to pilot agents
- Pilot agent uses peer and threat perception from cognitive model
- Pilot agent’s behavior tree checks for evade, disengage, bingo conditions
- Pilot agent’s behavior tree flies to intercept and eventually engage threat from task
- Pilot agent uses route finder to fly around SAM zones during ingress towards target

3.0 AFNES IADS MODEL

Algorithms and software from the Air Force mission level model were incorporated into the
AFSIM IADS model, including sensor modeling algorithms, the missile flyout code, and the
command chain logic in standard scenarios. Both the scenario data base (SDB) and type data base
(TDB) files for these scenarios have been translated into AFSIM format, and all Air Force
mission level model updates are then performance verified with the AFSIM IADS. This includes
comparisons of sensor vertical coverage diagram (VCD) plots for all sensor types in these
scenarios, as well as missile flyout comparisons for all surface-to-air missile (SAM) types. The
result is a continuous verification and validation (V&V) process to standard scenarios. Additional
information on the pedigree of the AFSIM IADS model can be requested by contacting the
AFSIM model manager.
Many mission-level constructive applications, including the AFSIM SAGE application, make use of one or more human-readable data files for input. These files are typically used to set system performance parameters and set initial conditions for the systems being modeled within a scenario of interest. Hand editing large scenario input files can be tedious and prone to error. In addition, most mission-level constructive applications output one or more data files representing the simulation results. Frequently, these files may include system positional histories and information on events that occurred during the simulation. Understanding the simulation output by simply reviewing the output files can be difficult, especially in the case of simulations with large numbers of entities and interactions.

### 4.0 SUMMARY

AFNES is a simulation environment that has been under development by Boeing under IR&D funds for more than 10 years. Under contract, Boeing delivered AFNES to the Air Force (specifically AFRL/RQQD) with unlimited government rights (including source code) in February 2013. AFRL/RQQD re-branded AFNES as the Advanced Framework for Simulation, Integration, and Modeling (AFSIM) and has started to distribute AFSIM within the Air Force and DoD, including DoD contractors. One of the key focus areas for AFSIM has been the simulated representation of IADS, and the utilization of verified and validated Air Force scenarios and threat representations distributed with the Air Force mission level model. AFSIM provides expanded Modeling & Simulation capabilities to support mission-level analysis studies related to global strike research activities.
Appendix A - AFSIM Concept of Operations

A1.0 Introduction

This section describes the architecture of the Advanced Framework for Simulation, Integration and Modeling (AFSIM), its applicability, and how users interact with applications that utilize the framework. It does not provide detailed descriptions of user inputs, algorithms or system models, but instead relegates this to other documents such as:

- The AFSIM Wiki, which serves as the reference manual for AFSIM commands.
- Appendix B, AFSIM Communications, Sensor and Jamming Equations, which provides the equations used by the standard system components in the framework.
- Appendix C, AFSIM Synthetic Aperture Radar Equations, which provides the equations used by the standard SAR model.
- Appendix D, AFSIM Electronic Warfare Overview, which defines the architecture of the standard EW model.

Abbreviated examples of user input will be provided to demonstrate general principles. These inputs are rendered as follows:

```plaintext
platform fighter-1 GENERIC_FIGHTER
...
end_platform
```

Using these abbreviated examples, users will be guided to the commands in the AFSIM Wiki which will provide more information.

a1.1 Applicability

AFSIM simulations are typically used to evaluate the performance of military systems in the context of a mission. To be successful, the framework must provide the capability to model the performance of the participants within the environment of the missions. In AFSIM, the individual participant is referred to as a platform, which in some simulations is called an entity. Platforms will be discussed in more detail later, but they represent things such as aircraft, satellites, missiles, ships, submarines, ground vehicles, structures and life-forms. The platform contains communications, sensors and weapons systems, and information and decision-making systems. These systems are used to gather, process and disseminate information, make command decisions, and carry out the commands. The framework, with its supplied system models as well other system models that may have been added, provides the capabilities to model the platforms participating in the simulation.

The properties of AFSIM platforms and subsystems are defined in human-readable input files. The files can be created with the AFSIM Integrated Development Environment (IDE) or any standard text editor. The IDE provides a context-sensitive text editor, the ability to run the simulation and view the results.
a1.2 Frameworks in General and AFSIM in Particular

A software framework can be roughly described as a cohesive collection of software components that aid in the development of applications. For instance, there are many frameworks for developing applications with graphical user interfaces that are independent of the operating system on which the application is used. For the user, the framework provides a consistent look and feel. For the programmer, it provides a consistent interface for interacting with the user and hides the system dependencies.

AFSIM is a framework for creating simulations typically associated with analyzing the performance of military systems in the context of a mission. The framework provides the ability to model the capabilities of the participants and to control the interaction of the participants as they move through space and time. The resulting simulations can be:

- Event-stepped or time-stepped.
- Constructive/non-interactive (the user invokes the simulation which then runs without further interaction) or interactive (the user or other simulation controls some aspects of the simulation).
- Non-real-time (faster or slower depending on the fidelity of the subsystem models) or real-time (constrained by some multiple of a real-time clock).

The AFSIM framework is written in ANSI/ISO standard C++ using modern OO programming techniques. It is designed to be extendable without changing the framework. For example, the framework contains many standard models for the following features:

- Movement systems
- Sensor systems
- Weapon systems and weapon effects
- Communication systems
- Information processing systems (trackers)
- Decision making systems (command and control, missile guidance, etc.)
- Antenna pattern models
- Atmospheric attenuation models
- Signal propagation models
- Clutter models
- Electronic warfare effect models

The framework allows other models of the above to be readily incorporated as needed. Once incorporated, the models are then as much a part of the framework as any of the standard models. The source for the other models is typically distributed separately from the framework and is then built with the application that requires them.

The framework also supports a ‘plug-in’ capability. Most of the model types defined above can be built as a plug-in and distributed separately from the main simulation executable. The plug-in can then be given to the user and used with an AFSIM simulation that has been compiled to use plug-ins. This allows capabilities to be added without the need to recompile the application.
a1.3 Limitations and Assumptions

There are very few constraints on the bounds of the mission. Memory is dynamically allocated so the number of participants is limited only by the amount of memory available and the amount of time one is willing to wait. WGS-84 coordinates are used to maintain the kinematic state of platforms so the mission may cover the whole world from underwater to space. Time is represented as a double precision variable so mission duration can be months or years.

Typically there is some level of aggregation in the definition of a platform, i.e., aircraft contain a pilot. For the purposes of simulation the pilot is melded into aircraft platform. When the aircraft moves, so does the pilot; when it is destroyed, so is the pilot.

There are no inherent limitations imposed by the framework on platforms or their subsystems with regard to their performance. Any platform can detect, communicate with, shoot or control any other platform. Any limitations are due either to the how the model is configured (user input) or limitations of a subsystem model. The fidelity of:

- Changing the parameters of the existing subsystem model to reflect the desired fidelity.
- Choosing a different subsystem model with the desired fidelity.
- Acquiring a different subsystem model with the desired fidelity.

For example, unless otherwise indicated, a sensor will attempt to detect all other platforms in the simulation. The user can make the assumption that it can distinguish friends from enemies and prevent detection attempts against friendly platforms, thus saving a lot of computation time. Another example is that a user may choose WSF_GEOMETRIC_SENSOR to model a radar system very simply (e.g. if the target is within 50 miles it can be seen). The user has assumed that this is a viable assumption, but if proven otherwise, the user can choose WSF_RADAR_SENSOR for more fidelity. The point is that the user is in charge of most limitations of this kind.

In order to complete a mission, platforms must have some sort of information on which to act. The truthfulness and accuracy of the information often governs the success of the mission. Assume a commander receives information that a target is located at a specific location and commands an asset to destroy the target. If the location had significant errors, or if the target did not really exist (it was a false report or the target had already been destroyed and the commander had not been informed), time and resources would have been wasted chasing a target that wasn’t there, thus potentially preventing the destruction of a real target. The framework imposes no inherent limitation on the viability of the information in a track. It can contain errors and does not have to correspond to a real platform.

a1.4 Requirements to Build and Run AFSIM Simulations

The standard applications (the framework and the standard unclassified applications released in a distribution) are written in ANSI/ISO Standard C++. The current list of supported software configurations is maintained on the AFSIM Wiki, but the standard applications are typically supported on:
- Redhat Enterprise Linux using the GCC compiler suites (32- or 64-bit memory model)
- Microsoft Windows using Microsoft Developer Studio C++ (32- or 64-bit memory model).

When the term ‘supported’ is employed, it means the standard applications will successfully compile and execute the standard test cases. Supported configurations are built and tested every day. Other variants of Linux generally work but are not officially supported. Other configurations that use the GCC compiler suite (e.g., MacOSX and MinGW) may work with some effort on the behalf of the user.

No additional software other than the above is required to build standard applications. The ‘Perl’ scripting language (www.perl.org) is required to run the automated tests. Users who will utilize the AFSIM plug-in capability must also acquire the ‘cmake’ utility (www.cmake.org).

The system requirements required to run an AFSIM-based application are wholly dependent on the application and the size of the problem being evaluated. The framework allocates all memory dynamically and adjusts itself to the size of the problem. Problems involving over 15,000 platforms have been run.
A2.0 Simulation Services

a2.1 Simulation Object

Every simulation created using the framework contains a simulation object (derived from the base class WsfSimulation). The simulation object has three major functions:

- Manage the flow of time
- Provide an event queue with methods to add and dispatch time-ordered events
- Provide for the orderly creation and destruction of platforms

The first two items are closely related. The event queue maintains a list of simulation events stored in order of increasing time. When requested to advance time forward to a specific time, the simulation will repeatedly look at the top event of the event queue. If the time is less than or equal to the requested time, the simulation will dispatch (execute) the event and remove it from the event queue. There are many types of events and new event types may be freely added.

The framework provides two standard simulation objects:

- A pure event-stepped simulation (WsfEventStepSimulation) in which time flows forward based on the time of the top event on the event queue.
- A frame-stepped simulation (WsfFrameStepSimulation) in which time flows forward in discrete steps. The advantage of a frame-stepped simulation is that it supports multi-threaded execution.

An object called a ‘clock source’ controls the ability of a simulation to run real-time or non-real-time. The purpose of the clock source is to provide a limiter on the flow of time. When attempting to move forward in time, the simulation will compare the time to the time provided by the clock source. If the simulation time is greater than the time provided by the clock source, the simulation will not move forward in time. The ‘null’ clock source is used for non-real-time simulations and does not limit the advance of time, thus allowing them to run as fast as possible. The real-time clock source simply limits time advancement requests to the elapsed time of a real-time clock, and is typically used in interactive simulations.

a2.2 Terrain and Line-of-Sight

Terrain is used by the framework to perform such actions as determining if objects involved in interactions are obscured by terrain, determining if an airborne object crashes into the ground, or to constrain a ground vehicle to the ground. The user input ‘terrain’ defines the locations of the terrain data as well as options that allow optimization of line-of-sight calculations. Acceptable forms of terrain data include:

- All levels of National Geospatial Agency (NGA) Digital Terrain Elevation Data (DTED). All data is accessible using the native directory structures. Different levels of DTED may be used simultaneously to cover different geographic regions.
- Arc-Info Float Grid format. Using this format, each terrain elevation “tile” consists of a binary file that contains the terrain elevation data and a header file that contains the geo-referencing information. When terrain elevation data are provided in this format,
the user may specify both the standard digital terrain model (DTM), as well as a digital surface model (DSM), or vegetation layer.

Technically, the height of the terrain is the height above the WGS-84 geoid. By default, the framework assumes the geoid and the ellipsoid are identical. The user can optionally provide information that describes the geoid undulation.

Line-of-sight functions are provided through the line-of-sight manager (User input: ‘line_of_sight_manager’). The manager will attempt to use cached results if the objects involved have not moved sufficiently since their last line-of-sight evaluation.

### a2.3 Observers, Event Logging, and External Simulation Interfaces

Internally, the framework implements a capability that allows simulation components to be notified of the occurrence of certain events (e.g.: turning a sensor on, sensor detection attempts, firing a weapon, sending a message assigning a task, etc.). Applications may register an ‘observer’ for any of the published events. Simulation components that perform the actions associated with the event will notify the simulation that the event has occurred. The simulation will in turn notify the observers of the event that it has occurred. The observer capability is used to implement capabilities such as event logging and external simulation interfaces without having to modify all sections of the code.

Two generic event logging capabilities are provided with the framework:

- ‘event_output’, which produces a human-readable file of information from user-selected events.
- ‘observer’, which allows the user to write scripts to perform any desired functions such as writing a custom output file or accumulating statistics. Many of the standard automated test cases use this capability to ensure that only expected events occur.

The framework also provides three external simulation interfaces:

- ‘dis_interface’, which allows an AFSIM simulation to participate in a Distributed Interactive Simulation (DIS). The interface also allows the production of a file that be displayed with the visualization tool VESPA.
- ‘hla_interface’, which allows an AFSIM simulation to participate in a High Level Architecture (HLA) exercise.
- ‘xio_interface’, which allows two or more AFSIM simulations to interact without the limitations of DIS and HLA.

The ‘event_output’, ‘dis_interface’, ‘hla_interface’ and ‘xio_interface’ are optional capabilities of the framework. The standard application ‘sage’ always includes ‘event_output’, ‘dis_interface’ and ‘xio_interface’, and will include the ‘hla_interface’ only if an acceptable HLA implementation is defined at build time.
A3.0 Supporting Capabilities

Before presenting platforms and platform subsystems, it is advantageous to present some of the supporting capabilities. Knowledge of these items is important to understanding the framework and how to use it.

a3.1 Scripts

The scripting language is an important part in the providing the flexibility of the framework. The scripting language is object-oriented and its syntax is similar to Java, Visual Basic/VBA or C#. It is not, however, a full-featured programming language. It does not provide a means to create custom objects, but rather is focused on accessing and modifying objects exposed by the simulation. Scripts are employed for:

- Implement tactics and doctrine (e.g., WSF_TASK_PROCESSOR)
- Data collection (e.g., ‘observer’)
- Dynamic configuration of objects based on mission options
- Simulation control

a3.2 Electromagnetic Framework

The electromagnetic (EM) framework is used by subsystem models (e.g., sensors, communications, and certain types of weapons) to model the transmission and reception of electromagnetic radiation. The EM framework provides a consistent set of objects that ensure complex interactions can be modeled accurately (e.g., passive RF, RF jammers). Note that while in theory the structure could be used for any type of EM interaction (radio frequency, visual, infrared), it is currently only fully employed for radio frequency.

a3.3 Electromagnetic Transmitters and Receivers

Transmitters and receivers for electromagnetic systems are defined using ‘transmitter’ and ‘receiver’ blocks embedded within the applicable sub-system model (sensors, communications, or weapon). For instance, a radar with a single beam would have a transmitter and receiver block:

```plaintext
sensor FIGHTER_RADAR WSF_RADAR_SENSOR
  transmitter
    ... transmitter commands ...
  end_transmitter
receiver
  ... receiver commands ...
end_receiver
end_sensor
```

The ‘transmitter’ block defines the characteristics of the emitted radiation (power, frequency, polarization, pulse width and pulse repetition frequency/interval and bandwidth for pulsed signals) and antenna pattern.

The ‘receiver’ block defines the characteristics of the receiver, such as the center operating frequency, bandwidth, operating polarization and antenna pattern.

When a system is turned on or off, the subsystem informs the simulation that the associated transmitters and receivers are being turned on or off. The simulation maintains a list of active transmitters and receivers ordered by frequency. The system also maintains a culled list of...
reciprocal objects on each transmitter and receiver with which it can interact with respect to frequency. This information is used by various subsystem models as follows:

- **WSF_ESM_SENSOR** uses the list of active transmitters within the frequency range of its receivers to determine if a particular transmitter can be detected. This eliminates the need to iterate over all transmitters to determine which can be detected.
- **WSF_RF_JAMMER** uses the culled list of jammer transmitters maintained on sensor or communications receivers to determine which jammer transmitters could possibly affect the receiver. This eliminates the need to iterate over all transmitters to determine which could affect the ability of the receiver to receive.

The lists are also updated whenever the frequency of an active transmitter or receiver is changed.

### a3.3.1 Antenna Patterns

Antenna patterns are attached to transmitter and receiver objects. A simple antenna pattern is a two-dimensional table that provides gain as a function of azimuth and elevation off the pointing angle of the antenna.

Antenna patterns are defined using the ‘antenna_pattern’ command outside of a subsystem definition, e.g.:

```plaintext
antenna_pattern EXAMPLE_RADAR_ANTENNA
    ... antenna_pattern commands
end_antenna_pattern
```

The antenna pattern is attached to a transmitter or receiver as follows:

```plaintext
sensor EXAMPLE_RADAR WSF_RADAR_SENSOR
transmitter
    antenna_pattern EXAMPLE_RADAR_ANTENNA
    ...
end_transmitter
...
end_sensor
```

The standard framework allows patterns to be defined using two-dimensional tables or using one of several algorithmic patterns. There are also several other optional models that provide additional pattern models.

A polarization- and frequency-dependent antenna pattern may be defined using the ‘antenna_pattern_table’ command in ‘transmitter’ and ‘receiver’ block.

### a3.3.2 Attenuation, Propagation and Clutter Models

Like antenna patterns, there are models of atmospheric attenuation, signal propagation and clutter. Atmospheric attenuation models are attached to ‘transmitter’ blocks in the same way as the antenna_pattern. For example:

```plaintext
transmitter
```
Clutter models are currently only applicable within the context of WSF_RADAR_SENSOR.

### a3.3.3 Electronic Warfare Effects

Transmitters associated with systems such as RF jammers (e.g., WSF_RF_JAMMER) may contain references that define the abilities of the transmitter to electronically attack receiver. Receivers associated with sensors or communications systems (e.g., WSF_RADARSENSOR or WSF_RADIO_TRANSCEIVER) may reference models that define the ability of the receiver to mitigate the effects of an electronic attack. For example:

```
weapon JAMMER WSF_RF_JAMMER
...
transmitter
  electronic_attack EA_TECHNIQUES
  ...
end_electronic_attack
...
end_transmitter
end_weapon

sensor RADAR WSF_RADARSENSOR
...
receiver
  electronic_protect EP_TECHNIQUES
  ...
end_electronic_protect
end_receiver
end_sensor
```

The EA_TECHNIQUES and EP_TECHNIQUES are previously defined by the ‘electronic_warfare’ command. The ‘electronic_warfare’ command contains a list of techniques for attacking or mitigating as defined by ‘electronic_warfare_technique’. The list of available techniques and their effects is more completely described in the document “AFSIM EW Architecture”.

### a3.4 Tracks and Track Management

The track is the fundamental object that represents the information known about another object such as a platform or a transmitter. The object may or may not be real, and the information may contain errors. Thus, it represents what a platform perceives about the other object. The track is the primary input to most of the decision-making process, and because it can represent flawed knowledge, poor decisions can be made.

There are two basic types of tracks:
- Raw tracks, which represent information from sensors or other off-board sources
- Local tracks, which represent the actual operating picture of a platform. Local tracks may be the result of filter and fusion processes involving multiple raw tracks.

The amount of information in a track depends on its source and filtering or fusion processes that may have been applied. A sensor may provide range and bearing, or it may only produce range bearing and elevation. It may also provide affiliation and type (e.g., IFF), measurement quality, etc.
A4.0 Platforms and Subsystems

a4.1 Platforms

The most visible object from the perspective of the user is the ‘platform’, which is in some simulations referred to as an ‘entity’. A platform can represent a person, an aircraft, spacecraft, naval vessel, building, etc. A platform object is primarily a container that contains the name, affiliation and commander of the platform, physical attributes such as its radar, optical and infrared signature, and definitions of propulsion, sensor, communication, weapon and processing subsystems.

In all but the most trivial of scenarios there are typically multiple instances of a given platform type (e.g., a homogeneous squadron of fighters). The user typically defines a file that contains the definition for a given platform type. In the framework this is done using the following:

```
platform_type GENERIC_FIGHTER WSF_PLATFORM
    ... platform commands ...
end_platform_type
```

This creates a new platform type, GENERIC_FIGHTER, that uses the definition of the built-in platform type WSF_PLATFORM as the starting point. This does not create an instance of a GENERIC_FIGHTER in the simulation, but rather creates a template from which to create instances. To create a squadron of three fighters the user would then enter the following:

```
platform fighter-1 GENERIC_FIGHTER
    side blue
    commander fighter-cmdr
    route
        ... route commands ...
    end_route
end_platform
platform fighter-2 GENERIC_FIGHTER
    ... side, commander and route commands for fighter-2 ...
end_platform
platform fighter-3 GENERIC_FIGHTER
    ... side, commander and route commands for fighter-3 ...
end_platform
```

Note that the using upper case for type names and lower case for instance names is a common convention.

a4.1.1 Signatures

The ‘signature’ of a platform defines its susceptibility to being seen by a sensor at a specified aspect angle. The framework provides the ability to define the signatures used by the standard sensor models provided by the framework. At a minimum, a signature definition is two-dimensional table that provides the susceptibility value as a function of the aspect angle (azimuth/elevation) of the viewing sensor with respect to the platform. All signatures have the ability to define a collection of tables that are indexed by a ‘configuration state’, where ‘configuration state’ represents something like bomb bay doors open or closed, or whether a
booster is burning. Radar signatures have additional levels of indexing to include the frequency and polarization of the incoming signal. For some cases it may be desirable to define signatures that are independent of aspect (i.e.: ‘cue-ball’ signatures). The framework provides a mechanism to do this using the constant command in lieu of a table definition.

A signature is defined by the presence of a radar_signature, infrared_signature or optical_signature block outside of a platform or platform_type block. For instance, to define the radar signature GENERIC_FIGHTER_RADAR_SIG:

```plaintext
radar_signature GENERIC_FIGHTER_RADAR_SIG
... Radar signature definition ...
end_radar_signature
```

The definition of the radar signature is then referenced in the definition of the platform type GENERIC_FIGHTER as follows:

```plaintext
platform_type GENERIC_FIGHTER WSF_PLATFORM
... radar_signature GENERIC_FIGHTER_RADAR_SIG ...
end_platform_type
```

Other signature types are defined and incorporated in the platform type in the same manner.

The configuration state of a platform may be changed dynamically using the AFSIM scripting interface. For instance, WSF_GUIDED_MOVER provides the ability for the user to define a script to be called when a rocket motor ignites or burns out, or when a staging event occurs. The user may provide definitions for these scripts that invoke the methods that change the signature state to reflect its new susceptibility.

There are other sensor models that have been integrated into the framework that may require more signature information than that provided by the framework. In such cases the additional signature information must be provided through the interface defined by the model. For instance, the Spectral Optical Sensor Model (SOSM) utilizes an infrared signature that is defined by wavenumber and not by band.

### 4.2 Platform Subsystems

Platform subsystems provide the models for the physical and logical operating capabilities of a platform. These will be described in the following sections.

#### 4.2.1 Mover

A mover model provides the means by which a platform moves through space-time. The mover is an optional subsystem; a platform without a mover is geographically fixed. As the simulation moves forward in time it invokes movers to cause it to update the platform’s position to reflect the current simulation time. When movers are invoked is dependent on the controlling simulation and the framework. The framework will ensure the position of a platform is current before involving it in any calculations.
There are many types of standard mover models, some of which are included below:

- **WSF_AIR_MOVER** – A route mover for air vehicle motion
- **WSF_GROUND_MOVER** – A route mover for a terrain-following ground vehicle
- **WSF_SURFACE_MOVER** – A route mover for a surface ship
- **WSF_SUBSURFACE_MOVER** – A route mover for a submersible vehicle
- **WSF_NORAD_SPACE_MOVER** – A mover for a platform in orbit about the Earth
- **WSF_GUIDED_MOVER** – A mover that is capable of representing a guided glide bomb or a single or multistate guided missile
- **WSF_TSPI_MOVER** – A mover that updates position based on Time Space Position Information (TSPI) data read from a text file

There are also several mover models available as add-on components that may not be available in all releases:

- **WSF_CSIMS_MOVER** – interface to the SIMS missile models
- **WSF_JAAM_MOVER** – interface to the JAAM missile models

### a4.2.2 Sensors

Sensor systems are used to sense the environment around platforms. There are several types of standard sensor models, some of which are:

- **WSF_ACOUSTIC_SENSOR** - Baseline acoustic sensor model
- **WSF_BEAM_DIRECTOR** - Sensor used to acquire very fine track, enabling firing of a high-energy laser
- **WSF_COMPOSITE_SENSOR** - A sensor composed of other sensors
- **WSF_EOIR_SENSOR** - Electro-Optical/Infrared (EOIR) sensor model
- **WSF_ESMSENSOR** - Baseline passive RF detection sensor
- **WSF_GEOMETRIC_SENSOR** - Baseline sensor based purely on geometry
- **WSF_IRST_SENSOR** - Baseline infrared search-and-track sensor
- **WSF_OPTICALSENSOR** - Baseline optical sensor model
- **WSF_OTH_RADAR_SENSOR** - Baseline Over-The-Horizon Backscatter (OTH-B) sky wave radar model
- **WSF_RADAR_SENSOR** - Baseline radar model
- **WSF_SAR_SENSOR** - Baseline synthetic aperture radar (SAR) model
- **WSF_SOSMSENSOR** - Interface to the Spectral Optical (IR) Sensing Model (SOSM)
- **WSF_SALRAM_SENSOR** - Interface to the Sea And Land Radar Model
- **WSF_SURFACE_WAVE_RADAR_SENSOR** - Over-the-horizon radar surface wave sensor model

### a4.2.3 Communications

Communications systems are used to transmit information from one platform to another. The framework offers three system models:
• WSFCOMM_TRANSCEIVER – Implements perfect or wired communications
• WSF_RADIO_TRANSCEIVER – Implements radio frequency communications
• WSF_JTIDS_TRANSCEIVER – Implements JTIDS/Link 16 communications

WSF_RADIO_TRANSCEIVER and WSF_JTIDS_TRANSCEIVER both use the electromagnetic subsystem and are subject to jamming.

a4.2.4 Weapons

Weapons systems are used to either lethally or non-lethally attack another platform or platform subsystem. A weapon system type is typically defined independently, and then instantiated on a platform type. The following example demonstrates how to create the definition of a surface-to-air missile (SAM) and its launcher. The weapon type WSF_EXPLICIT_WEAPON is the weapon used to launch a new platform that represents the missile.

```
platform_type SAM WSF_PLATFORM
   ... definitions of the SAM platform ...
end_platform

weapon SAM_LAUNCHER WSF_EXPLICIT_WEAPON
   Launched platform_type SAM
end_weapon

platform_type SAM_LAUNCHER WSF_PLATFORM
   ...
   weapon sam SAM_LAUNCHER
   end_weapon
end_platform_type
```

The standard framework also provides an RF jammer weapon (WSF_RF_JAMMER) and a laser weapon (WSF_LASER_WEAPON).

a4.2.5 Processors

Processors are used to implement the physical or logical processing capabilities of a platform. There are many processors available in the standard framework, among the most common being:

• WSF_TASK_PROCESSOR – a scriptable, finite-state machine for making command and control decisions about the tracks known to a platform.
• WSF_RIPR_PROCESSOR – a more advanced scriptable processor for making command and control decisions
• WSF_GUIDANCE_COMPUTER – implements a guidance computer for WSF_GUIDED_MOVER
• WSF_GROUND_TARGET_FUSE and WSF_AIR_TARGET_FUSE – implements a fusing mechanism for a weapon platform
• WSF_IMAGE_PROCESSOR – simulates analysis of ‘images’ produced by WSF_SAR_SENSOR and WSF_EOIR_SENSOR
• WSF _MESSAGE _PROCESSOR – a scriptable router and interpreter of messages
• WSF _TRACK _PROCESSOR – accepts tracks from on- and off-board sources and feeds them to the track manager. Also sends periodic updates of local tracks to other platforms

Depending on its implementation, a processor may be invoked periodically or as the result of some event such as the reception of a message or the expiration of a time interval.
Appendix B - AFSIM Communication, Sensor and Jamming Equations

B1.0 Overview

The purpose of this document is to describe the equations and algorithms used in the interactions between objects in AFSIM. This includes:

- Sensor interactions
- Communication interactions
- Disruption (jamming) interactions
B2.0 Common Radio Frequency Equations

AFSIM utilizes a common set of classes to encapsulate the components involved in radio frequency (RF) interactions (in reality, some features of these classes are also used for non-RF interactions, but that is not important here). The first section of the document will deal with the basics of signal transmission and reception. Subsequent sections of the document will deal with specific uses (radar, SAR, ESM, jamming, communications).

Ignoring the details of the processing of the received signal, RF interactions fall into two classes:

- Direct or one-way, i.e., an emitted signal going directly to a receiver
- Indirect or two-way: i.e., an emitted signal reflected from an object and then received

The computation of the received signal power can be broken into distinct steps:

- Emission from the transmitting antenna
- Propagation to the target or the receiver
- For indirect or two-way interactions
- Reflection from the target
- Propagation from the target to the receiver
- Reception by the receiving antenna

b2.1 Calculation of direct transmitted power

\[ P_x = P_{\text{peak}} \times DC \times \frac{G_x}{L_x} \]  
(RF.1)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Source</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( G_x )</td>
<td>transmitter</td>
<td>The gain of the transmitting antenna in the direction of the target object</td>
</tr>
<tr>
<td></td>
<td>antenna_pattern</td>
<td>(receiver or platform). This includes any electronic beam steering losses</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Equation RF.6)</td>
</tr>
<tr>
<td>( L_x )</td>
<td>transmitter</td>
<td>The internal losses in the transmitter between the power source and the</td>
</tr>
<tr>
<td></td>
<td>internal_loss</td>
<td>antenna</td>
</tr>
<tr>
<td>( DC )</td>
<td>transmitter</td>
<td>The user defined duty-cycle of the transmitter (default: 1.0, if not</td>
</tr>
<tr>
<td></td>
<td>duty_cycle</td>
<td>defined)</td>
</tr>
<tr>
<td>( P_{\text{peak}} )</td>
<td>transmitter</td>
<td>The peak power of the transmitter. This should be the power of a single</td>
</tr>
<tr>
<td></td>
<td>power</td>
<td>pulse.</td>
</tr>
<tr>
<td>( P_x )</td>
<td>Computed</td>
<td>The transmitted power.</td>
</tr>
</tbody>
</table>
b2.2 Propagation of the signal in free space

The propagation of a free space signal from the source (s) to the destination (d) is computed using the following equation. In a one-way interaction, ‘s’ and ‘d’ are the transmitter and receiver respectively (Equation RF.2b). In a two-way interaction, there are two propagation paths. The first is from the transmitter to the target (Equation RF.2c) and the second is from the target to receiver (Equation RF.2d).

\[
D_{sd} = P_s \times \frac{A_{sd}}{4\pi R_{sd}^2} \quad \text{General Form} \quad \text{(RF.2a)}
\]

\[
D_{sr} = P_s \times \frac{A_{sr}}{4\pi R_{sr}^2} \quad \text{Transmitter - to - receiver} \quad \text{(RF.2b)}
\]

\[
D_{xt} = P_s \times \frac{A_{xt}}{4\pi R_{xt}^2} \quad \text{Transmitter - to - target} \quad \text{(RF.2c)}
\]

\[
D_{xr} = P_t \times \frac{A_{xr}}{4\pi R_{xr}^2} \quad \text{Target - to - receiver} \quad \text{(RF.2d)}
\]

Table B-2. Free Space Propagation Variables

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Source</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A_{sd}</td>
<td>transmitter, attenuation_model</td>
<td>The fraction of the signal that remains after computing the effects of atmospheric attenuation while propagating the signal from the source (d) to the destination (d).</td>
</tr>
<tr>
<td>D_{sd}</td>
<td>Computed</td>
<td>The computed free space power density at the destination (s) that originated from the source (d).</td>
</tr>
<tr>
<td>P_s</td>
<td>Computed</td>
<td>The power emitted from the source (s). This will be either the transmitted power (Equation RF.1) or the reflected power from a target (Equation RF.3).</td>
</tr>
<tr>
<td>R_{sd}</td>
<td>Computed</td>
<td>The slant range from the source (s) to the destination (d).</td>
</tr>
</tbody>
</table>

b2.3 Reflecting a free space signal

A target that reflects a free space signal effectively creates a new ‘transmitting source’. The power of the source is simply the product of the signal density of the incoming signal times the effective area of the reflecting source. The reflector can be a platform (such as when performing a two-way radar interaction) or can be the surface of the earth (when performing clutter calculations). The reflected power can then be propagated to a receiver by application of equation RF.2.

\[
P_t = D_{sr} \times \sigma_t \quad \text{(RF.3)}
\]
Table B-3. Free Space Reflected Signal Variables

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Source</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D_{xt}$</td>
<td>Equation RF.2c</td>
<td>The power density at the target ($t$) of the signal that originated from the transmitter ($x$).</td>
</tr>
<tr>
<td>$P_t$</td>
<td>Computed</td>
<td>The power created by the reflection of the incoming signal off of the target.</td>
</tr>
<tr>
<td>$\sigma_t$</td>
<td>radar_signature of the target platform</td>
<td>The radar cross section of the target.</td>
</tr>
</tbody>
</table>

b2.4 Reception of a free space signal

RF.4a is used for direct, one-way (communications, passive RF and jamming). RF.4b is used two-way (Radar, SAR).

\[
P_r = D_{xr} \times \frac{\lambda^2}{4\pi} \times \frac{G_r}{L_r} \times F_{BW} \times F_{POL}
\]
One-way, Transmitter - to - receiver (RF.4a)

\[
P_r = D_{tr} \times \frac{\lambda^2}{4\pi} \times \frac{G_r}{L_r} \times F_{40}
\]
Two-way, Target - to - receiver (RF.4b)

Table B-4. Free Space Received Power Variables

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Source</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_{BW}$</td>
<td>See section 2.5</td>
<td>The fraction of the received signal that is admitted, accounting for possible mismatches in the frequency/bandwidth of the transmitted and the frequency/bandwidth of the receiver. Note: this is not incorporated for radar interactions because it is assumed that the transmitter and receiver are matched.</td>
</tr>
<tr>
<td>$F_{POL}$</td>
<td>transmitter polarization receiver polarization polarization_effects antenna_pattern</td>
<td>The fraction of the received signal that is admitted, accounting for possible mismatches in the polarization of the transmitter and the receiver. Note: This is not incorporated for radar interactions because it is assumed that the transmitter and receiver are matched.</td>
</tr>
<tr>
<td>$F_{40}$</td>
<td>transmitter propagation_model</td>
<td>The pattern propagation factor. This accounts for the constructive/destructive interference between the direct and indirect signal paths. Note: This is current only implemented for radar interactions.</td>
</tr>
<tr>
<td>$D_{xr}$</td>
<td>Equation RF.2b</td>
<td>The power density at the receiver of the signal that originated from the transmitter.</td>
</tr>
<tr>
<td>$D_{tr}$</td>
<td>Equation RF.2d</td>
<td>The power density at the receiver of the signal that was reflected from the target.</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>transmitter wavelength</td>
<td>The wavelength of the transmitted signal.</td>
</tr>
<tr>
<td>$G_r$</td>
<td>receiver antenna_pattern</td>
<td>The gain of the receiving antenna in the direction of the target object (receiver or platform). This includes any effects of electronic beam steering (Equation RF.6).</td>
</tr>
<tr>
<td>$L_r$</td>
<td>receiver internal_loss</td>
<td>The internal losses in the receiver between the output of the antenna and the receiver.</td>
</tr>
<tr>
<td>$P_r$</td>
<td>Computed</td>
<td>The received power.</td>
</tr>
</tbody>
</table>
b2.5 Bandwidth ratio

The factor $F_{BW}$ is used to account for the fact that the frequency spectrum of the transmitter may not match the tuning band of the receiver. It is the fraction of the transmitter spectrum that is within the tuning band of the receiver.

$$F_{ul} = F_i - \frac{1}{2}B_t \quad \text{Lower frequency of transmitted spectrum}$$
$$F_{uu} = F_i + \frac{1}{2}B_t \quad \text{Upper frequency of transmitted spectrum}$$
$$F_{rl} = F_r - \frac{1}{2}B_r \quad \text{Lower tuning frequency of the receiver}$$
$$F_{ru} = F_r + \frac{1}{2}B_r \quad \text{Upper tuning frequency of the receiver}$$

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Source</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B_r$</td>
<td>receiver bandwidth</td>
<td>The bandwidth of the receiver.</td>
</tr>
<tr>
<td>$B_t$</td>
<td>transmitter bandwidth</td>
<td>The bandwidth of the transmitter.</td>
</tr>
<tr>
<td>$F_r$</td>
<td>receiver frequency</td>
<td>The center frequency of the range of frequencies the receiver can receive.</td>
</tr>
<tr>
<td>$F_t$</td>
<td>transmitter frequency</td>
<td>The center frequency of the transmitter frequency spectrum</td>
</tr>
</tbody>
</table>

The resulting value of $F_{BW}$ depends on the relationship of the upper and lower frequencies of the transmitter and receiver.

$$F_{BW} = 0 \quad \text{if } F_{su} \leq F_{rl}$$
$$F_{BW} = 0 \quad \text{if } F_{sl} \geq F_{ru}$$
$$F_{BW} = \min \left( \frac{\min(F_{su}, F_{ru}) - \max(F_{sl}, F_{rl})}{F_{su} - F_{sl}}, 1.0 \right) \quad \text{(RF.5)}$$

b2.6 Receiver noise power

The following definitions apply to the computation of receiver noise power:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Source</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k$</td>
<td>Internal constant</td>
<td>Boltzmann’s constant (1.3806505E-23 J/deg-K)</td>
</tr>
<tr>
<td>$B$</td>
<td>receiver bandwidth</td>
<td>The bandwidth of the receiver. If the bandwidth was not specified AND if the transmitter is pulsed, the bandwidth will be computed as (1 / pulse_width) (i.e.: A matched filter will be assumed).</td>
</tr>
</tbody>
</table>
The noise power will be computed using the following process. The value from the first step whose conditions for use are satisfied will be used:

1. If `noise_power` was specified, used the defined value.

2. If the bandwidth cannot be determined, use the value of -160 dBW.

3. If `noise_figure` was specified and both `antenna_ohmic_loss` and `receive_line_loss` were omitted, compute the noise power as:

   \[ N = k \times T_0 \times B \times NF \quad \text{(RF.6a)} \]


   Noise temperature due to the antenna (\( T_{\text{ant}} \) = sky temperature due to the antenna pointing angle):

   \[ T_a = T_0 + (0.876 \times T_{\text{ant}} - 254.0) / \text{antenna_ohmic_loss} \quad \text{(RF.6b)} \]

   Noise temperature contribution due to receive line loss:

   \[ T_r = T_0 \times (\text{receive_line_loss} - 1.0) \quad \text{(RF.6c)} \]

   Noise temperature contribution due to the receiver:

   \[ T_r = T_0 \times (\text{noise_figure} - 1.0) \quad \text{(RF.6d)} \]

   Total system temperature:

   \[ T_s = T_a + T_r + (\text{receive_line_loss} \times T_r) \quad \text{(RF.6e)} \]

   Noise power:

   \[ N = k \times T_s \times B \quad \text{(RF.6f)} \]

### b2.7 Antenna Gain Patterns

Each transmitter and receiver has associated with it an antenna gain pattern. Antenna patterns are created using the global `antenna_pattern` command. An antenna pattern is attached to a transmitter or receiver by using the `antenna_pattern` command inside the `transmitter` and

Distribution A: Approved For Public Release (Case #:88ABW-2014-5041)
receiver block. If an antenna pattern has not been selected for a transmitter or receiver, a uniform gain of 1.0 will be assumed.

The gain pattern is a function of azimuth and elevation with respect to the pattern origin (typically the bore sight or pointing angle). For a given interaction, the azimuth and elevation of the point of interest with respect to the pattern origin is computed.

Antenna gain patterns may be represented in several ways:

- A rectangular table which provides the gain as a function of azimuth and elevation.
- A table.
- A uniform (constant) pattern.
- A circular \( \sin(x)/x \) pattern.
- A rectangular \( \sin(x)/x \) pattern.
- A cosecant pattern.
- A GENAP pattern (GENAP is a subset of the functionality provided by generalized antenna pattern routine found in the government TRAMS model).

Collections of tables may be used to form a composite pattern of polarization and frequency.

The gain of an electronically steered beam can be optionally modified to include the effects of pointing the beam at an angle off the normal of the array. This capability is enabled by using the `electronic_beam_steering` command in the `transmitter` or `receiver`. The following equation is used:

\[
G = G_0 \times \cos^n(\theta) \quad \text{Equation RF.6}
\]

### Table B-7. Antenna Gain Variables

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Source</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( G_0 )</td>
<td>antenna_pattern</td>
<td>The unmodified gain of the antenna when looking at the point of interest.</td>
</tr>
<tr>
<td>( G )</td>
<td>Computed</td>
<td>The gain, modified to include the effects of electronic beam steering.</td>
</tr>
<tr>
<td>( \theta )</td>
<td>Computed</td>
<td>The angle between the normal to the antenna face and the vector to the point of interest.</td>
</tr>
<tr>
<td>( N )</td>
<td>electronic_beam_steering_loss_exponent</td>
<td>An optional exponent to reflect the amount of degradation of the gain as the beam is moved away from the normal of the antenna face.</td>
</tr>
</tbody>
</table>

### b2.8 Atmospheric Attenuation

Computation of atmospheric attenuation is enabled by the presence of the `atmospheric_attenuation` command inside the `transmitter` block.
There are currently two models available. These were extracted from Air Force models and are currently only applicable to ground-based systems (the tables assume the emitter is on the ground).

- An atmospheric absorption model written by L.V. Blake, Naval Research Laboratory. This is based on a family of 42 attenuation curves for frequencies between 100 MHz and 10 GHz and elevation angles between 0 and 10 degrees. The curves are flat beyond 300 nautical miles. These tables were published in 'Radar Systems Analysis’, Section 15.1, David K. Barton, Artech Publishing.

- A collection of precomputed tables that are valid for frequencies in the range 100 MHz to 18 GHz and 27 GHz to 40 GHz. Frequencies less than 100 MHz will assume 100 MHz. Frequencies between 18 GHz and 27 GHz and above 40 GHz will use a very computationally-intensive method to determine the attenuation and should be avoided.

There is another model in development based on the International Telecommunications Union (ITU) Recommendation ITU-R P.676. That implementation will work for air and surface platforms and support a wider range of frequencies.

b2.9 Propagation Algorithms

Computation of propagation effects (other than atmospheric attenuation) is enabled by the presence of the propagation_model command inside the transmitter block.

This currently supports one model:

- fast_multipath - An implementation of the method defined in 'Radar Range Performance Analysis’, Lamont V. Blake, 1986, Artech House, Inc. It computes the effects of constructive or destructive interference due to the specular reflection of the signal off of a round, rough Earth. Two factors can be supplied to define the properties of the surface at the reflection point.

b2.10 Clutter Algorithms

AFSIM currently has a very limited ability to represent clutter. The use of clutter is enabled by the presence of the clutter_model command in the receiver block. At the current time the only option is to utilize a clutter table, and it has not been validated.
B3.0 Radar Sensor (WSF_RADAR_SENSOR)

The AFSIM radar model effectively computes the power of a single pulse (or a continuous waveform) and then computes the effect of integrating multiple pulses.

b3.1 Calculation of Received Power

Applying equations RF.1 through RF.4, the following is used to calculate the received power from a single pulse (or a continuous waveform). Note that this does not including jamming. That is handled in a separate step.

\[
P_r = D_{tr} \times \frac{\lambda^2}{4\pi} \times \frac{G_r}{L_r} \times F_{40}
\]

From RF.4b

\[
P_r = P_i \times \frac{A_{tr}}{4\pi R_{tr}^2} \times \frac{\lambda^2}{4\pi} \times \frac{G_r}{L_r} \times F_{40}
\]

From RF.2d

\[
P_r = D_{st} \times \sigma_t \times \frac{A_{tr}}{4\pi R_{sr}^2} \times \frac{\lambda^2}{4\pi} \times \frac{G_r}{L_r} \times F_{40}
\]

From RF.3b

\[
P_r = P_t \times DC \times \frac{G_r}{L_r} \times \frac{A_{st}}{4\pi R_{sr}^2} \times \sigma_t \times \frac{A_{tr}}{4\pi R_{tr}^2} \times \frac{\lambda^2}{4\pi} \times \frac{G_r}{L_r} \times F_{40}
\]

From RF.1 (Radar.1)

b3.2 Signal Processing and Detection

The processed signal is computed as:

\[
S = P_r \times PCR \times G_t \times AF
\]

(Radar.2)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Source</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AF</td>
<td>adjustment_factor</td>
<td>A general adjustment factor that can be used to account for other constant effects that are not provided by the model.</td>
</tr>
<tr>
<td>Gi</td>
<td>integration_gain</td>
<td>The gain due to the integration of multiple pulses. Note: This is computed internally if swerling_case is specified.</td>
</tr>
<tr>
<td>PCR</td>
<td>transmitter pulse_compression_ratio</td>
<td>The pulse compression ratio</td>
</tr>
<tr>
<td>Pr</td>
<td>Equation Radar.1</td>
<td>The received power</td>
</tr>
<tr>
<td>S</td>
<td>Computed</td>
<td>The processed power</td>
</tr>
</tbody>
</table>

The signal to noise is computed as:

\[
SN = \frac{S}{N + C + J}
\]

(Radar.3)
Table B-9. Signal and Noise Variables

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Source</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>Receiver clutter_model</td>
<td>The clutter power</td>
</tr>
<tr>
<td>J</td>
<td>Equation Jam.1</td>
<td>The incident jammer power. This is computed as the sum of the incident power on the radar receiver at the time of the detection interaction.</td>
</tr>
<tr>
<td>N</td>
<td>Equation RF.6</td>
<td>The receiver noise power</td>
</tr>
<tr>
<td>S</td>
<td>Equation Radar.2</td>
<td>The processed power.</td>
</tr>
<tr>
<td>SN</td>
<td>Computed</td>
<td>The signal-to-noise (or interference) ratio.</td>
</tr>
</tbody>
</table>

The detection of the target is determined by one of two mechanisms. A simplistic binary detector may be used by specifying a `detection_threshold`. A successful detection is declared if the signal-to-noise exceeds this threshold.

A Marcum-Swerling detector may also be used, producing a probability of detection for a given signal-to-noise ratio. A successful detection is declared if the computed probability of detection exceeds the required probability of detection. This detector is selected by using the `swerling_case`, `number_of_pulses_integrated`, `probability_of_false_alarm` and `detector_law` commands.
B4.0 Passive RF Sensor (WSF_ESM_SENSOR)

Passive RF calculation (ESM, RWR) utilize the one-way equation.

The ‘r’ subscript values are for the passive RF receiver and the ‘x’ subscript values are for the sensor, jammer or communications transmitter. The expanded equation is as follows:

$$
P_r = D_{sr} \times \frac{\lambda^2}{4\pi} \times \frac{G_r}{L_r} \times F_{BW} \times F_{POL}$$  \hspace{1cm} \text{From RF.4a}

$$
= P_x \times \frac{A_{sr}}{4\pi R_{sr}^2} \times \frac{\lambda^2}{4\pi} \times \frac{G_r}{L_r} \times F_{BW} \times F_{POL}$$  \hspace{1cm} \text{From RF.2b}

$$
= P_{peak} \times DC \times \frac{G_s}{L_x} \times \frac{A_{sr}}{4\pi R_{sr}^2} \times \frac{\lambda^2}{4\pi} \times \frac{G_r}{L_r} \times F_{BW} \times F_{POL}$$  \hspace{1cm} \text{From RF.1 (ESM.1)}

The signal-to-noise is computed as:

$$
SN = \frac{P_r}{N} \quad \text{(ESM.2)}
$$

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Source</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>Equation RF.6</td>
<td>The receiver noise power</td>
</tr>
<tr>
<td>$P_r$</td>
<td>Equation ESM.1</td>
<td>The processed power.</td>
</tr>
<tr>
<td>SN</td>
<td>Computed</td>
<td>The signal-to-noise (or interference) ratio.</td>
</tr>
</tbody>
</table>

A successful detection will be declared if $SN$ exceeds the threshold defined by:

- The value of **pulsed_detection_threshold** if the transmitted signal is pulsed
- The value of **continuous_detection_threshold** if the transmitted signal was non-pulsed
- The value of **detection_threshold** if neither of the above thresholds was specified
B5.0 SAR Sensor (WSF_SAR_SENSOR)

SAR calculations are an extension of the radar calculations.

b5.1 Required Collection Time

The equation used to compute the time required to collect an image of the desired resolution is:

\[
T_{ot} = \frac{KR}{2d_AV_G \sin(\alpha) \cos(\delta)} \quad \text{(SAR.1)}
\]

Table B-11. SAR Collection Time Variables

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Source</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(d_A)</td>
<td>Computed</td>
<td>The desired azimuth resolution</td>
</tr>
<tr>
<td>(K)</td>
<td>doppler_overcollect_ratio</td>
<td>The overcollect ratio (default 1.0)</td>
</tr>
<tr>
<td>(R_s)</td>
<td>Computed</td>
<td>The slant range from the sensor to the image center</td>
</tr>
<tr>
<td>(V_G)</td>
<td>Computed</td>
<td>The ground speed of the sensing platform.</td>
</tr>
<tr>
<td>(\lambda)</td>
<td>transmitter frequency –or– wavelength</td>
<td>The frequency of the transmitted signal</td>
</tr>
<tr>
<td>(\alpha)</td>
<td>Computed</td>
<td>The azimuth angle between the ground track of the sensing platform and the vector to the image center.</td>
</tr>
<tr>
<td>(\delta)</td>
<td>Computed</td>
<td>The azimuth angle between the ground and the vector to the image center.</td>
</tr>
</tbody>
</table>
B6.0 RF Jammer (WSF_RF_JAMMER)

Jamming calculations utilize the one-way equation where the transmitter is the jammer and the receiver is a radar or communications receiver. Jamming calculations take place at the time a radar detection or communication attempt occurs. AFSIM will sum the power for every possible jammer than can affect the output (i.e.: if there is in-band power that would affect the receiver).

The ‘r’ subscript values are for the sensor or communications receiver and the ‘x’ values are for the jamming transmitter. The expanded equation is a follows:

\[
P_r = \frac{D_{sr}}{4\pi L_r} \times G_r \times F_{BW} \times F_{POL}
\]

\[
= \frac{P_x \times A_{sr}}{4\pi R_{sr}^2} \times \frac{\lambda^2}{4\pi} \times \frac{G_r}{L_r} \times F_{BW} \times F_{POL}
\]

\[
= P_{peak} \times DC \times \frac{G_x}{L_x} \times \frac{A_{sr}}{4\pi R_{sr}^2} \times \frac{\lambda^2}{4\pi} \times \frac{G_r}{L_r} \times F_{BW} \times F_{POL}
\]

From RF.4a
From RF.2b
From RF.1 (Jam.1)
B7.0 Communications (WSF_RADIO_TRANSCEIVER)

Communications calculations use the one-way equation. The ‘r’ subscript values are for the communications receiver and the ‘x’ values are for the communications transmitter. The expanded equation is as follows:

\[
P_r = \frac{D_{sr}}{4\pi} \times \frac{\lambda^2}{L_r} \times G_r \times F_{BW} \times F_{POL}
\]

\[
= P_x \times \frac{A_{sr}}{4\pi R_{sr}^2} \times \frac{\lambda^2}{4\pi} \times \frac{G_r}{L_r} \times F_{BW} \times F_{POL}
\]

\[
= P_{peak} \times DC \times \frac{G_r}{L_r} \times \frac{A_{sr}}{4\pi R_{sr}^2} \times \frac{\lambda^2}{4\pi} \times \frac{G_r}{L_r} \times F_{BW} \times F_{POL}
\]

From RF.1 (Comm.1)

The signal-to-noise ratio is computed as:

\[
SN = \frac{P_r}{N + J}
\]

(Comm.2)

Table B-12. Communications Signal and Noise Variables

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Source</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>J</td>
<td>Equation Jam.1</td>
<td>The incident jammer power. This is computed as the sum of the incident power on the receiver at the time of the interaction.</td>
</tr>
<tr>
<td>N</td>
<td>Equation RF.6</td>
<td>The receiver noise power</td>
</tr>
<tr>
<td>P_r</td>
<td>Equation Comm.1</td>
<td>The processed power.</td>
</tr>
<tr>
<td>SN</td>
<td>Computed</td>
<td>The signal-to-noise (or interference) ratio.</td>
</tr>
</tbody>
</table>

The communications attempt will be declared successful if \( SN \) exceeds the \textit{detection\_threshold} for the receiver.
B8.0 IRST Sensor (WSFIRSTSENSOR)

b8.1 Computing the target irradiance

Determine the background radiance. This includes a relatively simply capability to include the effects of looking up against the sky or down at the ground.

Compute the contrast radiant intensity;

$$I_c = I_s - \frac{L_{bkg}}{A_{proj}}$$

Table B-13. IRST Contrast Radiant Intensity Variables

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Source</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I_s$</td>
<td>platform infrared_signature</td>
<td>The source radiant intensity (infrared radiant intensity) of the target</td>
</tr>
<tr>
<td>$L_{bkg}$</td>
<td></td>
<td>The background radiant intensity</td>
</tr>
<tr>
<td>$A_{proj}$</td>
<td>platform optical_signature</td>
<td>The projected area of the target as seen by the sensor.</td>
</tr>
<tr>
<td>$I_c$</td>
<td>Computed</td>
<td>The contrast radiant intensity of the target</td>
</tr>
</tbody>
</table>

Compute the atmospheric transmittance (the fraction of the signal that remains after propagation along the path):

Compute the effective target irradiance (sometimes known as CEI)

$$E_{eff} = \frac{t \times I_c}{R^2}$$

b8.2 Adjusting for installation effects

Sensors are often mounted behind a window, which will mask regions from the field of view, or otherwise reduce the signal. This masking or signal reduction is collectively called ‘installation effects’, and is accounted for by the use of an antenna_pattern command in the receiver block (While there is no ‘antenna’ in an infrared sensor, it is being treated as such for convenience). The command should refer to an antenna gain pattern where the gain (or more possibly, the loss) represents a factor, by which the effective target irradiance should be modified to account for installation effects, i.e.:

$$E'_{eff} = E_{eff} \times G \quad \text{(IRST.1)}$$

where $G$ is the ‘antenna gain’ in the direction of interest. Setting the gain to a very small value in the regions that are outside the window effectively makes targets in that region undetectable.

b8.3 Computing the probability of detection

The probability of detection is computed using the following equation:
\[ SN = \frac{E_{\text{eff}}}{NEI} \]
\[ \beta = SN - S_{\text{thresh}} \]
\[ P_d = 1 - Q(\beta) \]

### Table B- 14. IRST Probability of Detection Variables

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Source</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( E_{\text{eff}} )</td>
<td>Equation IRST.1</td>
<td>The effective target irradiance</td>
</tr>
<tr>
<td>( NEI )</td>
<td>noise_equivalent_irradiance</td>
<td>The equivalent irradiance of the noise of the sensor.</td>
</tr>
<tr>
<td>( P_d )</td>
<td>Computed</td>
<td>The probability of detection</td>
</tr>
<tr>
<td>( Q(\beta) )</td>
<td></td>
<td>The Gaussian probability function (see the ‘Handbook of Mathematic Functions’, Abramowitz and Stegun, equation 26.2.5)</td>
</tr>
<tr>
<td>( SN )</td>
<td>Computed</td>
<td>Signal-to-noise ratio</td>
</tr>
<tr>
<td>( S_{\text{thresh}} )</td>
<td>detection_threshold</td>
<td>The detection threshold</td>
</tr>
</tbody>
</table>
Appendix C– AFSIM Synthetic Aperture Radar Equations

C1.0 The Radar Range Equation

The following equation is the standard equation used by AFSIM to compute the received power from a single pulse of a radio frequency signal that is transmitted, reflects off an object and is then received. This does not assume the transmitter and receiver are co-located, and does not account for any additional signal processing techniques such as pulse compression or integration of multiple pulses:

\[
P_r = P_{\text{peak}} (DC) \left( G_t \right) \left( \frac{A_{xt}}{4\pi R_{xt}^2} \right) \sigma \left( \frac{A_{tr}}{4\pi R_{tr}^2} \right) \left( \frac{\lambda^2}{4\pi} \right) \left( \frac{G_r}{L_r} \right) \left( F_{40} F_{BW} F_{POL} \right)
\]

where:

Table C-1. Radar Range Equation Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>AFSIM source</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P_{\text{peak}} )</td>
<td>transmitter power</td>
<td>The peak transmitted power</td>
</tr>
<tr>
<td>DC</td>
<td>transmitter duty_cycle</td>
<td>The user defined duty-cycle of the transmitter (default: 1.0, if not defined)</td>
</tr>
<tr>
<td>( \lambda )</td>
<td>transmitter frequency or wavelength</td>
<td>The wavelength of the radiated signal</td>
</tr>
<tr>
<td>( G_t, G_r )</td>
<td>transmitter / receiver antenna_pattern</td>
<td>The gain of the transmitter and receiver antennas</td>
</tr>
<tr>
<td>( L_x, L_r )</td>
<td>transmitter / receiver internal_loss</td>
<td>The internal losses within the transmitter and receiver</td>
</tr>
<tr>
<td>( A_{xt}, A_{tr} )</td>
<td>transmitter attenuation_model</td>
<td>The one-way atmospheric attenuation factor (0..1) computed by the attenuation mode.</td>
</tr>
<tr>
<td>( R_{xt}, R_{tr} )</td>
<td>Computed</td>
<td>The range from the transmitter-to-target and target-to-receiver.</td>
</tr>
<tr>
<td>( \sigma )</td>
<td></td>
<td>The radar cross section of the target (which could be the target or a ‘resolution cell’).</td>
</tr>
<tr>
<td>( F_{40} )</td>
<td>transmitter propagation_model</td>
<td>The pattern propagation factor that accounts for the constructive and destructive interference between the direct and indirect reflections.</td>
</tr>
<tr>
<td>( F_{BW} )</td>
<td></td>
<td>The factor that accounts for bandwidth mismatches between the bandwidth of the transmitted signal and the bandwidth of the receiver. This is primarily for interactions between radar transmitters and passive RF receivers, or jammers and radar receivers. It is not intended to capture the effects of non-ideal matched filters in a system.</td>
</tr>
<tr>
<td>( F_{POL} )</td>
<td>transmitter / receiver polarization</td>
<td>The factor that accounts for the polarization mismatch of the transmitted signal and the receive antenna.</td>
</tr>
</tbody>
</table>

For a SAR, the transmitter and receiver are co-located, so \( R = R_{xt} = R_{tr} \) and \( A = A_{xt} = A_{tr} \). Also, the gain of the transmit and receive antennas will be assumed identical, so \( G = G_t = G_r \). Furthermore, the following will be assumed:
• There are no indirect signals to interfere with the main signal. Therefore $F_{40} = 1$.
• The bandwidth of the receiver has been established to capture the full bandwidth of the transmitted signal. Therefore, $F_{BW} = 1$.
• The polarization of the received signal is the same as the polarization of the transmitted signal. Therefore, $F_{POL} = 1$.
• If we also define the total atmospheric attenuation loss as:

$$L_{atm} = \frac{1}{A_{xt}A_{tr}} = \frac{1}{A^2}$$

With the above assumptions, we get the familiar equation for power received from a single pulse, where the transmitter and receiver are co-located:

$$P_r = P_{peak} \frac{G^2 \lambda^2 \sigma}{(4\pi)^3 R^4 L_x L_r L_{atm}}$$

This is the same as equation (1) in reference 1, noting that:

$$L_{radar} = L_x L_r$$

and that equation (8) has been used to replace $A_e$:

$$A_e = \frac{G_A \lambda^2}{4\pi}$$

Further note that reference 1 goes on to represent the effective aperture as a product of the actual aperture area multiplied by an ‘aperture efficiency’. We will not perform that step here and assume that any aperture efficiency has been represented in the AFSIM antenna_pattern.

The received signal must compete with the noise present within the system. AFSIM computes noise power using the following:

**Table C-2. Receiver Noise Variables**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>AFSIM source</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k$</td>
<td>Internal constant</td>
<td>Boltzmann’s constant (1.3806505E-23 J/deg-K)</td>
</tr>
<tr>
<td>$B_N$</td>
<td>receiver bandwidth</td>
<td>The bandwidth of the receiver. If the bandwidth was not specified AND if the transmitter pulse_width is specified, the bandwidth will be computed as (1 / pulse_width) (i.e.: a matched filter will be assumed.</td>
</tr>
<tr>
<td>$F_N$</td>
<td>receiver noise figure</td>
<td>The receiver noise figure (default 1.0)</td>
</tr>
<tr>
<td>$N$</td>
<td>Computed</td>
<td>The noise power.</td>
</tr>
<tr>
<td>$T_0$</td>
<td>Internal constant</td>
<td>The standard temperature (290 deg-K)</td>
</tr>
</tbody>
</table>

The receiver noise at the antenna port is computed as:

$$N = kT_0 B_N F_N$$

(Note: Section 2.6 in reference 2 describes other forms of computing the noise power, but these are primarily for surface-based systems.)
The signal-to-noise ratio for a single pulse at the antenna port is then:

\[ SNR_{ant} = \frac{P_r}{N} = \frac{P_{peak}}{N} = \frac{G^2 \lambda^2 \sigma}{(4\pi)^3 R^4 L_x L_r L_{atm} k T_0 B_N F_N} \]

(This is the same as equation (5) in reference 1, with the substitutions noted above.)

A SAR utilizes two signal processing techniques to increase the effective SNR in the image.

- \( G_a \) = SNR gain due to azimuth processing (coherent pulse integration).
- \( G_r \) = SNR gain due to range processing (pulse compression)

This will result in the signal-to-noise ratio a target within the image to be:

\[ SNR_{image} = SNR_{ant} G_a G_r = \frac{P_r}{N} = \frac{P_{peak}}{N} = \frac{G^2 \lambda^2 \sigma}{(4\pi)^3 R^4 L_x L_r L_{atm} k T_0 B_N F_N} G_a G_r \]

(This is the same as equation (11) in reference 1, with the substitutions noted above.)

### C2.0 Azimuth Processing Gain (Coherent Pulse Integration)

The creation of a SAR image involves the collection of a large number of pulses coherently over some duration of time that is suitable for producing an image of the desired quality.

<table>
<thead>
<tr>
<th>Variable</th>
<th>AFSIM source</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f_p )</td>
<td>transmitter pulse_repetition_frequency</td>
<td>The pulse repetition frequency</td>
</tr>
<tr>
<td>( K_a )</td>
<td>doppler_filter_broadening_factor</td>
<td></td>
</tr>
<tr>
<td>( K_d )</td>
<td>doppler_foldover_margin_factor</td>
<td></td>
</tr>
<tr>
<td>( t_D )</td>
<td>dwell_time or computed</td>
<td>The dwell time, or image collection time.</td>
</tr>
<tr>
<td>( \delta_a )</td>
<td>resolution or computed</td>
<td>The desired azimuth resolution.</td>
</tr>
<tr>
<td>( V )</td>
<td></td>
<td>The vehicle velocity vector.</td>
</tr>
<tr>
<td>( \theta_{sq} )</td>
<td>computed</td>
<td>The ‘squint angle’, defined as the angle between the velocity vector and the line-of-sight vector to the center of the image area.</td>
</tr>
<tr>
<td>( n_{image} )</td>
<td>Computed</td>
<td>The total number of pulses collected in forming the image</td>
</tr>
</tbody>
</table>

Equation (5) from Reference 3 is used to compute the dwell time from the desired cross range/azimuth resolution:

\[ t_D = \frac{\lambda K_a R}{2 V \delta_a \sin \theta_{sq}} \]

Note that AFSIM lets the user specify either the desired resolution or dwell time. In the latter case, AFSIM will use the above equation to solve for the achievable resolution given the dwell time.
The azimuth gain is the total number of pulses collected, which is just the collection time times the pulse repetition frequency:

\[ G_a = n_{image} = t_D f_p = \frac{\lambda K_a R f_p}{2V_\delta_a \sin \theta_{sq}} \]

### C2.0 Range Processing Gain (Pulse Compression)

#### Table C-4. Range Processing Gain Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>AFSIM source</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \tau_u )</td>
<td>transmitter</td>
<td>The uncompressed pulse width.</td>
</tr>
<tr>
<td>( \tau_c )</td>
<td></td>
<td>The pulse compression ratio.</td>
</tr>
<tr>
<td>( \frac{\tau_u}{\tau_c} )</td>
<td>transmitter</td>
<td>The pulse compression ratio.</td>
</tr>
<tr>
<td>( \text{pulse_compression_ratio} )</td>
<td></td>
<td>The pulse compression ratio.</td>
</tr>
</tbody>
</table>

The range processing gain due to pulse compression is:

\[ G_r = \frac{\tau_u}{\tau_c} \]

### C3.0 Various Forms Of The Signal-To-Noise Equation

Substituting the results for \( G_a \) and \( G_r \) into the equation for \( SNR_{image} \):

\[
SNR_{image} = P_{\text{peak}} \left( \frac{G^2 \lambda^2 \sigma}{4\pi^3 R^4 L_x L_I L_{atm}} \right) \frac{1}{k T_0 B_N F_N} \frac{G_a G_r}{G_a} \frac{\tau_c}{\tau_u} \frac{K_a f_p}{2V_\delta_a \sin \theta_{sq}} \frac{\tau_u}{\tau_c}
\]

This is the form used by AFSIM to compute the return from an object with a radar cross section of \( \sigma \). This could be a target or a resolution cell.

Additional forms of the equation are often seen in the literature. The remainder of this section will show how the above equation is equivalent.

In the case of a matched filter:

\[ B_N = \frac{1}{\tau_c} \]

Substituting:

\[
SNR_{image} = P_{\text{peak}} \left( \frac{G^2 \lambda^3 \sigma}{2 \pi^3 R^3 L_x L_I L_{atm}} \right) \frac{1}{k T_0 B_N F_N} \frac{G_a f_p}{2V_\delta_a \sin \theta_{sq}} \frac{\tau_c}{\tau_u} \frac{K_a}{\tau_c}
\]

\[
SNR_{image} = P_{\text{avg}} \left( \frac{G^2 \lambda^3 \sigma}{2 \pi^3 R^3 L_x L_I L_{atm}} \right) \frac{1}{k T_0 F_N} \frac{G_a f_p}{2V_\delta_a \sin \theta_{sq}} \frac{\tau_c}{\tau_u} \frac{K_a}{\tau_c}
\]

Where average power is defined to be:

\[ P_{avg} = P_{\text{peak}} \tau_u f_p \]

53
One form of interest is when the target is a bare resolution cell (i.e.: the ground). This is sometimes called the ‘clutter-to-noise ratio’, or CNR. Equation (23) of reference 1 defines the area of the resolution cell as:

\[ \sigma = \sigma_0 \delta_r \delta_{rg} = \sigma_0 \delta_a \frac{\delta_r}{\cos \psi_g} \]

where:

Table C-5. Resolution Cell Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>AFSIM source</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \sigma^0 )</td>
<td>backscatter_coefficient</td>
<td>The backscatter coefficient</td>
</tr>
<tr>
<td>( \delta_r )</td>
<td></td>
<td>Range resolution (as computed from the effective pulse width)</td>
</tr>
<tr>
<td>( \delta_{rg} )</td>
<td></td>
<td>Range resolution in the ground plane</td>
</tr>
<tr>
<td>( \psi_g )</td>
<td></td>
<td>Grazing angle. The angle between the line-of-sight vector and the plane tangent to the surface at the point being viewed.</td>
</tr>
</tbody>
</table>

Substituting:

\[ SNR_{image} = P_{avg} \frac{G^2 \lambda^3}{(4\pi)^3 R^3 L_x L_{atm}} \sigma_0 \delta_a \frac{\delta_r}{\cos \psi_g} \frac{1}{k T_0 F_N} \frac{2V \delta_a \sin \theta_{sq}}{K_a} \]

which is basically equivalent to the myriad forms presented in Appendix B of reference 1 (however they always assumed broadside collection, so \( \sin \theta_{sq} \) was always 1).

C4.0 Creation of AFSIM Pseudo-Images

AFSIM does not produce true images, but rather produces pseudo-images that indicate the objects that are in the image, the number of resolution cells (pixels) occupied by the object, and the intensity of the object.

- The user cues the system to the desired location and turns the system on. The model constructs a list of the targets that could potentially be in the image. The target list will encompass targets that are slightly outside the image region in order to account for the fact that a target may move into the image.
- At periodic intervals (defined by ‘frame_time’, default of 1 second), the model computes and accumulates data for each of the targets from step 1. The results of these detection results will be accumulated, much as a SAR accumulates pulses. If the target is obscured by terrain during a given sample, it will not have any contributing pulses defined during that interval.
- At some point, the SAR will be turned off. At that point the model will take the accumulated results and produce the pseudo-image (WsfImage) and send a message containing the image (WsfImageMessage) to those who have subscribed.

The following variables will be used in the following section:
### Table C-6. Pseudo-Image Generation Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>AFSIM source</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_F$</td>
<td>frame_time</td>
<td>The update interval between samples when forming the image.</td>
</tr>
<tr>
<td>$n_{actuat}$</td>
<td></td>
<td>The actual number of pulses integrated. This may be different from $n_{image}$ if the sensor was turned off before or after the required time.</td>
</tr>
<tr>
<td>$t_{sample}$</td>
<td></td>
<td>The length of the sample. This will be $t_F$ for all but the final sample.</td>
</tr>
<tr>
<td>$n_{sample}$</td>
<td></td>
<td>The number of pulses received during a sample.</td>
</tr>
<tr>
<td>$P_{sample}$</td>
<td></td>
<td>The received signal from a specific target during a sample.</td>
</tr>
<tr>
<td>$NP_{sample}$</td>
<td></td>
<td>The number of resolution cells (pixels) covered by a specific target during a sample.</td>
</tr>
<tr>
<td>$\sigma_{opt}$</td>
<td>optical_signature</td>
<td>The optical signature of the target.</td>
</tr>
<tr>
<td>$P_{sum}$</td>
<td></td>
<td>The sum of the sampled received signals for a specific target.</td>
</tr>
<tr>
<td>$NP_{sum}$</td>
<td></td>
<td>The sum of the sampled pixel counts for a specific target.</td>
</tr>
<tr>
<td>$N_{seen}$</td>
<td></td>
<td>The number of samples in which a specific target was visible (not obscured by the terrain).</td>
</tr>
<tr>
<td>$P_{ref}$</td>
<td></td>
<td>The reference signal that corresponds to a zero intensity in the output image. This would typically be minimum clutter-to-noise ratio.</td>
</tr>
<tr>
<td>$P_{range}$</td>
<td></td>
<td>A normalizing value used to scale the received signals into a range of $[0..1]$.</td>
</tr>
<tr>
<td>$CNR$</td>
<td></td>
<td>The expected signal-to-noise ratio of a return from single resolution cell.</td>
</tr>
<tr>
<td>$CNR_{min}$</td>
<td>detection_threshold</td>
<td>The minimum acceptable value for CNR for an image to be declared</td>
</tr>
</tbody>
</table>

Step 1 computes the anticipated dwell time ($t_D$) and the number of pulses to be collected ($n_{image}$). In addition, it calculates the anticipated value of $CNR$

For each samples in step 2, the number of pulses received during the sample interval is:

$$n_{sample} = f_p t_{sample}$$

The number of resolution cells (pixels) occupied by the target for a given sample is projected area of the target (optical cross section) divided by the size of a resolution cell:

$$NP_{sample} = \frac{\sigma_{opt}}{\delta_a \delta_r}$$

The received power per resolution cell from the target during the sample interval is:

$$P_{sample} = \frac{SNR_{image} n_{sample}}{N n_{image} NP_{sample}}$$

Note that the noise has been removed from the accumulation. This must be done to account for the possibility that in some samples the target may not be visible, or that the actual dwell time may be longer or shorter what was initially computed. The other terms account for the fact that
the internal routine that calculates $SNR_{image}$ computes the return for the entire target for the expected dwell time.

For each sample of a target in which the target is not obscured by terrain, the following is performed:

\[
N_{seen} = N_{seen} + 1 \\
P_{sum} = P_{sum} + P_{sample} \\
NP_{sum} = NP_{sum} + NP_{sample}
\]

In step 3, the achieved clutter-to-noise ratio must be computed. It is done at this point because the actual number of pulses collected is now known (the user may choose to turn the system off before or after the time required).

\[
CNR_{actual} = CNR \frac{n_{actual}}{n_{image}}
\]

If $CNR_{actual}$ is greater than or equal to $CNR_{min}$, the image will be declared acceptable and will contain the targets as processed below. If image is declared unacceptable, the image will be produced with no targets.

The reference signal level will be defined to be:

\[
P_{ref} = \frac{CNR_{actual}}{N}
\]

If the image is declared acceptable, the following will be produced for each target:

The number of pixels (resolution cells) occupied by the target. This will just be the average of the pixel counts from each sample where there target was not obscured by terrain:

\[
NP = \frac{NP_{sum}}{N_{seen}}
\]

The intensity of the pixel is then computed. The integrated return from a resolution cell (aka, clutter cell) represents the ‘zero’ intensity, or the return that will return a pixel value of zero.

\[
I = \frac{P_{sum} - P_{ref}}{P_{range}}
\]

A value less than zero is clamped to zero, while values greater than one are clamped to one.

C5.0 References

2. Johnson, Jeffery, “AFSIM Communications, Sensor and Jamming Equations”, Boeing

56

Distribution A: Approved For Public Release (Case #:88ABW-2014-5041)
Appendix D – AFSIM Electronic Warfare Overview

D1.0 Electronic Warfare Architecture

The Electronic Warfare (EW) capability in AFSIM provides for the effect(s) of an EW technique from existing data, lower-level models (engineering models, engagement models, etc.) to be modeled without having to capture a lot of high-fidelity information (i.e. pulse level modeling). This results in primarily data/table driven capabilities, and the use of equations where behavior is well defined, documented, and proven to follow specific guidelines. The benefits of this level of modeling provide improved runtime, and allows for less detail on the many interactions of different effects associated with techniques on different systems to be evaluated using single/few model definitions. The cons are a possible loss in fidelity and some edge cases that may not always be captured properly.

The general coding approach to implementing the EW techniques in AFSIM was to follow an Object-Oriented (OO) approach/framework allowing for easier addition/update of new EW effects, while adding some complexity on the positioning of data and methods within the architecture. The EW architecture was further divided into Electronic Attack (EA) and Electronic Protect (EP) classes, with ES being considered in another part of the AFSIM architecture. Each of the EA and EP classes has multiple techniques, each with their own predefined and user-defined characteristics known as effects that the technique supplies as shown in Figure D-1.

![Figure D-1. EW Techniques Architecture](image)

Distribution A: Approved For Public Release (Case #:88ABW-2014-5041)
EA techniques are associated with transmitting systems within the AFSIM definition of a system as shown in Figure D-2. Likewise, the EP techniques are associated with receiving systems as shown in Figure D-3. The transmitter can then deliver the EA techniques to a receiver upon a successful interaction between the transmitter and receiver. The interaction between the EA & EP techniques and their associated effects are accomplished through user mappings of the EA & EP interactions and occur during a transmitter-receiver interaction as depicted in Figure D-3. Although Figure D-2 depicts a sensor, the interaction of EW techniques is also pertinent to communications systems in AFSIM, as they share the same base receiver functionality.

![Figure D-2. Jamming System Architecture in AFSIM Showing EA Techniques](image-url)
Figure D-3. Radar System Architecture in AFSIM Showing EP techniques

Figure D-4. Mapping of EA and EP Interactions in the AFSIM EW Architecture
D2.0 Electronic Attack Effects

EA specific effects in AFSIM that are defined are as follows with a short description:

- **WSF_COMM_EFFECT**
  - Induce communication effects
- **WSF_COVER_PULSE_EFFECT**
  - Induce cover pulse effects with probabilities
- **WSF_FALSE_TARGET_EFFECT**
  - False-Target effect with track creation
  - Pulse type effect
- **WSF_POL_MOD_EFFECT**
  - Derives from the WSF_SLC_DEGRADE_EFFECT with the addition of some data to model a polarization modulation technique
  - Induces a degradation on the EP SLC effect
- **WSF_POWER_EFFECT**
  - Jammer gain/degrade effect
  - Base type for most effects
- **WSF_PULSE_EFFECT**
  - Pulse level effects
  - Base type for other pulse level base types
- **WSF_RADIUS_EFFECT**
  - Target/Jammer/Radar trio radius effects
- **WSF_REPEATER_EFFECT**
  - Application of repeater jamming effects
- **WSF_RPJ_EFFECT**
  - Random Pulse Jamming/ Modulation (RPJ/RPM) effect
  - Pulse type effect
- **WSF_SIMPLE_FT_EFFECT**
  - Simple False-Target effect w/o track creation
  - Pulse type effect
- **WSF_SLC_DEGRADE_EFFECT**
  - Extra SLC degradation factors
- **WSF_TRACK_EFFECT**
  - Induce track error effect(s)

These EA effects also utilize inheritance from base classes within AFSIM, as depicted in Figure D-5, that are also able to have their input commands used within the inheriting class from a command input level.
D3.0 Electronic Protect Effects

EP specific effects in AFSIM that are defined are as follows with a short description:

- **WSF_AGILITY_EFFECT**
  - Frequency and/or mode agility/diversity effects
- **WSF_COMM_EFFECT**
  - Mitigate communication effects
- **WSF_POWER_EFFECT**
  - Jammer gain/degrade effect
  - Base type for most effects
- **WSF_PULSE_EFFECT**
  - Pulse level effects
  - Base type for other pulse type effects
- **WSF_PULSE_SUPPRESS_EFFECT**
  - Pulse Suppression effect for pulse type EA effects
  - Pulse type effect
- **WSF_SLB_EFFECT**
  - Sidelobe Blanker effects
- **WSF_SLC_EFFECT**
  - Sidelobe Canceller effects
These EP techniques also use inheritance from base classes within AFSIM, as depicted in Figure D-6, that are also able to have their input commands executed within the inheriting class from a command input level.

**Figure D-6. Hierarchy of Base Type Effects for EP**

**D4.0 EW Interaction Overview**

As previously described, the EW effects in AFSIM include EA effects and EP effects that are applied during a transmitter-receiver or transmitter-target-receiver interaction as defined by this effects behavior in software and user modifiable inputs. During this interaction, the interaction data for the EW are applied by using the unmitigated EA effect, then applying any mitigating EP effects to the EA effect. This process is then repeated until all EA effects on all techniques are applied along with mitigating EP effects, and aggregated into a single data set that is used to modify the interaction between the transmitter-receiver and transmitter-target-receiver. The aggregation of this EW effects is further defined below for more insight into how AFSIM is applying these effects and is graphically depicted in Figure D-7.

**d4.1 Electronic Attack Effect Coherency**

Jamming power within an interaction calculation is calculated using general AFSIM Electromagnetic calculations. Each EA effect possesses one or more of the coherency types on
the individual effect. As the effects are applied, the types of effect coherencies encountered are summed, and at the end of all the EW effects calculations the jamming power is divided into three types of jamming power (non-coherent, non-coherent-pulse, coherent) based on the EA effects coherencies encountered during the application of the effects. The coherency types available are defined in the table below along with the jamming power type it is summed into.

### Table D-1. Effect Coherency Types

<table>
<thead>
<tr>
<th>Effect Coherency Types</th>
<th>Description</th>
<th>Mapped Jamming Power Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>Coherency not specified for the given effect. Assumes Non-Coherent</td>
<td>Noise</td>
</tr>
<tr>
<td>Non-Coherent</td>
<td>Waveform is non-coherent with the transmit and/or expected receive waveform. Assumed continuous noise type waveform in most basic sense.</td>
<td>Noise</td>
</tr>
<tr>
<td>Non-Coherent Pulse</td>
<td>Waveform is pulsed and is non-coherent with the transmit and/or expected receive waveform. Assumed pulsed noise type waveform in most basic sense.</td>
<td>Pulsed Noise</td>
</tr>
<tr>
<td>Coherent</td>
<td>Waveform is coherent with the transmit and/or expected receive waveform. Assumed to be closely representing the signal in most basic sense.</td>
<td>Coherent</td>
</tr>
<tr>
<td>Coherent-Pulse</td>
<td>Waveform is pulse and is coherent with the transmit and/or expected receive waveform. Assumed to be closely representing the pulsed signal in most basic sense.</td>
<td>Coherent</td>
</tr>
</tbody>
</table>

### Table D-2. Jamming Power Types

<table>
<thead>
<tr>
<th>Jamming Power Type</th>
<th>Description</th>
<th>Mapped Effect Coherency Type(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Noise</td>
<td>Jamming induced power that acts like noise power to a receiver.</td>
<td>None &amp; Non-Coherent</td>
</tr>
<tr>
<td>Pulsed-Noise</td>
<td>Pulse jamming powered that acts like noise to a receiver.</td>
<td>Non-Coherent Pulse</td>
</tr>
<tr>
<td>Coherent</td>
<td>Coherent (continuous and/or pulsed) jamming power that acts like a signal to a receiver.</td>
<td>Coherent &amp; Coherent Pulse</td>
</tr>
</tbody>
</table>

Within most interactions the signal-to-interference (S/I) ratio is calculated using the signal power divided by the noise power + clutter power + jammer power. The jammer power used for the interference is the noise + pulse (non-coherent) jammer power.
d4.2 EW Effects Interaction Variables

The following EW effects variable structure is defined for each of the three types of jamming power as well as a separate signal, track and message effect structures. The following tables summarize these two structures and associated variables:

**Table D-3. Jamming Effects Variable Structure**

<table>
<thead>
<tr>
<th>Power Effect Variable</th>
<th>Description</th>
<th>Aggregation Type(s)</th>
<th>Default / Undefined Value</th>
<th>Modifying Effect(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blanking Factor</td>
<td>Jamming blanking factor (e.g., sidelobe blanker)</td>
<td>Multiplicative</td>
<td>1.0</td>
<td>WSF_SLB_EFFECT</td>
</tr>
<tr>
<td>Cancellation Factor</td>
<td>Jamming cancellation factor (e.g., sidelobe canceller)</td>
<td>Minimum</td>
<td>1.0</td>
<td>WSF_SLC_EFFECT, WSF_SLC_DEGRADE_EFFECT</td>
</tr>
<tr>
<td>Modulation Factor</td>
<td>Jamming processing/modulation type factor, not to physical jamming power factor.</td>
<td>Multiplicative</td>
<td>1.0</td>
<td>WSF_POWER_EFFECT</td>
</tr>
<tr>
<td>Jamming power Factor</td>
<td>Jamming physical power factor.</td>
<td>Multiplicative</td>
<td>1.0</td>
<td>WSF_POWER_EFFECT, WSFCOVER_PULSE_EFFECT</td>
</tr>
<tr>
<td>J/X Factor</td>
<td>Alternate jamming processing/modulation type that has a Jamming-to-Signal/Noise dependency.</td>
<td>Multiplicative</td>
<td>1.0</td>
<td>WSF_POWER_EFFECT</td>
</tr>
<tr>
<td>Target Protection Flag</td>
<td>Flag to specify whether or not jamming power will be allowed to interact with the receiver for a given target or not.</td>
<td>Undefined</td>
<td>undefined</td>
<td>Base Effect</td>
</tr>
<tr>
<td>Pulse Suppression Factor</td>
<td>Pulse type jamming suppression factor.</td>
<td>Multiplicative</td>
<td>1.0</td>
<td>WSFPULSE_SUPPRESS_EFFECT</td>
</tr>
<tr>
<td>Radius Factor</td>
<td>Factor that evaluates the position of the target wrt jammer location to apply a user input factor.</td>
<td>Multiplicative</td>
<td>1.0</td>
<td>WSF_RADIUS_EFFECT</td>
</tr>
<tr>
<td>Repeater</td>
<td>Physical jamming</td>
<td>Multiplicative</td>
<td>1.0</td>
<td>WSF_REPEATER_EFFECT</td>
</tr>
<tr>
<td>Jamming Factor</td>
<td>power factor dependent upon repeater behavior defined.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RPJ Factor</td>
<td>Random pulse jamming factor.</td>
<td>Multiplicative</td>
<td>1.0</td>
<td>WSF_RPJ_EFFECT</td>
</tr>
</tbody>
</table>

**Table D-4. Signal Effects Variable Structure**

<table>
<thead>
<tr>
<th>Signal Effect Variable</th>
<th>Description</th>
<th>Aggregation Type(s)</th>
<th>Default/Undefined Value</th>
<th>Modifying Effect(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal Power Factor</td>
<td>Signal power factor.</td>
<td>Multiplicative</td>
<td>1.0</td>
<td>WSF_POWER_EFFECT</td>
</tr>
<tr>
<td>Receiver Noise Power Factor</td>
<td>Receiver noise power factor.</td>
<td>Multiplicative</td>
<td>1.0</td>
<td>WSF_POWER_EFFECT</td>
</tr>
</tbody>
</table>

**Table D-5. Track Effects Variable Structure**

<table>
<thead>
<tr>
<th>Track Effect Variable</th>
<th>Description</th>
<th>Aggregation Type(s)</th>
<th>Default/Undefined Value</th>
<th>Modifying Effect(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Azimuth Error</td>
<td>Track azimuth error.</td>
<td>Maximum (EA) / Minimum (EP)</td>
<td>0.0</td>
<td>WSF_TRACK_EFFECT</td>
</tr>
<tr>
<td>Elevation Error</td>
<td>Track elevation error.</td>
<td>Maximum (EA) / Minimum (EP)</td>
<td>0.0</td>
<td>WSF_TRACK_EFFECT</td>
</tr>
<tr>
<td>Range Error</td>
<td>Track range error.</td>
<td>Maximum (EA) / Minimum (EP)</td>
<td>0.0</td>
<td>WSF_TRACK_EFFECT</td>
</tr>
<tr>
<td>Velocity Error</td>
<td>Track velocity error.</td>
<td>Maximum (EA) / Minimum (EP)</td>
<td>0.0</td>
<td>WSF_TRACK_EFFECT</td>
</tr>
<tr>
<td>Track Drop/Maintain Flag</td>
<td>Track drop/maintain flag</td>
<td>Undefindeed Boolean</td>
<td>undefined</td>
<td>WSF_TRACK_EFFECT, WSF_SLB_EFFECT (target blanking)</td>
</tr>
</tbody>
</table>
Table D-6. Message Effects Variable Structure

<table>
<thead>
<tr>
<th>Track Effect Variable</th>
<th>Description</th>
<th>Aggregation Type(s)</th>
<th>Default/Undefined Value</th>
<th>Modifying Effect(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit Error Rate (BER)</td>
<td>BER for communications device to use.</td>
<td>Maximum (EA) / Minimum (EP)</td>
<td>0.0</td>
<td>WSF_COMM_EFFECT</td>
</tr>
<tr>
<td>Message Drop/Maintain Flag</td>
<td>Message drop/maintain flag</td>
<td>Undefined Boolean</td>
<td>undefined</td>
<td>WSF_COMM_EFFECT</td>
</tr>
</tbody>
</table>

d4.3 Aggregation Types

The aggregation types given in the table below are used to aggregate (i.e., roll-up) the individual EW effect values into the interaction value to be used by the interaction to apply any EW related effects to the target detection, tracking process and/or message as applicable. All aggregation is done in standard units (i.e., multiplication is the same as addition in dB space.)

Table D-7. Aggregation Types

<table>
<thead>
<tr>
<th>Aggregation Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum</td>
<td>The maximum of the interaction value and current effect value being applied is taken and used as the interaction value.</td>
</tr>
<tr>
<td>Minimum</td>
<td>The minimum of the interaction value and current effect value being applied is taken and used as the interaction value.</td>
</tr>
<tr>
<td>Additive</td>
<td>The addition of the interaction value and current effect value being applied is used as the interaction value.</td>
</tr>
<tr>
<td>Multiplicative</td>
<td>The multiplied product of the interaction value and current effect value being applied is used as the interaction value.</td>
</tr>
<tr>
<td>Boolean</td>
<td>A true/false (i.e., two-state) flag that can be toggled based on the current value and logging of the effect.</td>
</tr>
<tr>
<td>Undefined Boolean</td>
<td>Similar to the Boolean aggregation type, except an undefined state along with the true/false (i.e., three-state) is available as a value. This type can be toggled from undefined (its most common default state) to true/false (i.e., defined) and toggled between the three states thereafter based on the current value and logic of the effect.</td>
</tr>
</tbody>
</table>
Figure D-7. AFSIM EW Interaction Flowchart
# List of Acronyms, Abbreviations, and Symbols

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFNES</td>
<td>Analytic Framework For Network Enabled Systems</td>
</tr>
<tr>
<td>AFSIM</td>
<td>Advanced Framework for Simulation, Integration, and Modeling</td>
</tr>
<tr>
<td>AFRL</td>
<td>Air Force Research Laboratory</td>
</tr>
<tr>
<td>ANSI</td>
<td>American National Standards Institute</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>AVTAS</td>
<td>Aerospace Vehicles Technology Assessment &amp; Simulation</td>
</tr>
<tr>
<td>CC</td>
<td>Conventional Campaign</td>
</tr>
<tr>
<td>CNR</td>
<td>Clutter to Noise Ratio</td>
</tr>
<tr>
<td>DEM</td>
<td>Digital Elevation Model</td>
</tr>
<tr>
<td>DIS</td>
<td>Distributed Interactive Simulation</td>
</tr>
<tr>
<td>DLL</td>
<td>Dynamic Link Libraries</td>
</tr>
<tr>
<td>DOE</td>
<td>Design Of Experiments</td>
</tr>
<tr>
<td>DTED</td>
<td>Digital Terrain Elevation Data</td>
</tr>
<tr>
<td>EM</td>
<td>Electromagnetic</td>
</tr>
<tr>
<td>ENU</td>
<td>East, North, Up</td>
</tr>
<tr>
<td>EA</td>
<td>Electronic Attack</td>
</tr>
<tr>
<td>EP</td>
<td>Electronic Protect</td>
</tr>
<tr>
<td>ESM</td>
<td>Electronic Support Measure</td>
</tr>
<tr>
<td>EW</td>
<td>Electronic Warfare</td>
</tr>
<tr>
<td>Geo</td>
<td>Geo-referencing</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HLA</td>
<td>High Level Architecture</td>
</tr>
<tr>
<td>HTML</td>
<td>HyperText Markup Language</td>
</tr>
<tr>
<td>IADS</td>
<td>Integrated Air Defense Systems</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
<tr>
<td>IR&amp;D</td>
<td>Internal Research and Development</td>
</tr>
<tr>
<td>ISO</td>
<td>International Organization for Standardization</td>
</tr>
<tr>
<td>ITU</td>
<td>International Telecommunications Union</td>
</tr>
<tr>
<td>M&amp;S</td>
<td>Modeling and Simulation</td>
</tr>
<tr>
<td>MCO</td>
<td>Major Combat Operations</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>NCO</td>
<td>Network Centric Operations</td>
</tr>
<tr>
<td>NED</td>
<td>North, East, Down</td>
</tr>
<tr>
<td>NGA</td>
<td>National Geospatial-Intelligence Agency</td>
</tr>
<tr>
<td>OSG</td>
<td>Open Scene Graph</td>
</tr>
<tr>
<td>OO</td>
<td>Object Oriented</td>
</tr>
<tr>
<td>PIANO</td>
<td>Parametric Integrated Analysis of Objectives</td>
</tr>
<tr>
<td>RF</td>
<td>Radio Frequency</td>
</tr>
<tr>
<td>RIPR</td>
<td>Reactive Integrated Planning Architecture</td>
</tr>
<tr>
<td>RWR</td>
<td>Radar Warning Receiver</td>
</tr>
<tr>
<td>SAGE</td>
<td>Simulation of Autonomously Generated Entities</td>
</tr>
<tr>
<td>SAM</td>
<td>Surface-to-Air Missile</td>
</tr>
<tr>
<td>SAR</td>
<td>Synthetic Aperture Radar</td>
</tr>
<tr>
<td>SDB</td>
<td>Scenario Data Base</td>
</tr>
<tr>
<td>SIMS</td>
<td>Standard Industry Missile Simulator</td>
</tr>
<tr>
<td>SNR</td>
<td>Signal to Noise Ratio</td>
</tr>
<tr>
<td>SO</td>
<td>Shared Objects</td>
</tr>
<tr>
<td>STK</td>
<td>System Tool Kit</td>
</tr>
<tr>
<td>TDB</td>
<td>Type Data Base</td>
</tr>
<tr>
<td>TIFF</td>
<td>Tagged Image File Format</td>
</tr>
<tr>
<td>UCAV</td>
<td>Unmanned Combat Air Vehicle</td>
</tr>
<tr>
<td>USAF</td>
<td>United States Air Force</td>
</tr>
<tr>
<td>USGS</td>
<td>United States Geological Survey</td>
</tr>
<tr>
<td>V&amp;V</td>
<td>Verification &amp; Validation</td>
</tr>
<tr>
<td>VCD</td>
<td>Vertical Coverage Diagram</td>
</tr>
<tr>
<td>VESPA</td>
<td>Visual Environment for Scenario Preparation and Analysis</td>
</tr>
<tr>
<td>VMAP</td>
<td>Vector Map</td>
</tr>
<tr>
<td>VTK</td>
<td>VESPA ToolKit</td>
</tr>
<tr>
<td>WGS</td>
<td>World Geodetic System</td>
</tr>
</tbody>
</table>