Copy No. _____

Defence Research and Development Canada | Recherche et développement pour la défense Canada

DEFENCE R&D DÉFENSE

AIMSsim version 2.3.4

User Manual

Oliver Schoenborn, Patrick Lachance and Nima Bahramifarid
CAE Professional Services

CAE Professional Services
1135 Innovation Drive, Suite 300
Ottawa, Ontario K2K 3G7

Project Manager: O. Schoenborn, 613-247-0342
Contract Number: W7711-047904/TOR/001
Contract Scientific Authority: J. Crebolder, 902-426-3100 x296

Defence R&D Canada – Atlantic

Contract Report
DRDC Atlantic CR 2007-300
January 2008

Canada

AIMSsim version 2.3.4

User Manual

Oliver Schoenborn, Patrick Lachance and Nima Bahramifarid
CAE Professional Services

CAE Professional Services
1135 Innovation Dr., Suite 300
Ottawa, ON
K2K 3G7

Project Manager: O. Schoenborn, 613-247-0342

Contract number: W7711-047904/TOR/001

Contract Scientific Authority: J. Crebolder, 902-426-3100 x296

Defence R&D Canada – Atlantic

Contract Report

DRDC Atlantic CR 2007-300
January 2008

Author

Oliver Schoenborn

Approved by

Original signed by Jacquelyn Crebolder

Jacquelyn Crebolder

Approved for release by

Original signed by James L. Kennedy

James L. Kennedy

Abstract

This user manual provides an overview of the software functionality developed to support the empirical investigation of a simulated user interface for an Advanced Integrated Multi-sensor Surveillance (**AIMS**) system (formerly known as the Enhanced Low-Light Level Visible and Infrared Surveillance System – **ELVISS**). The AIMS system is an electro-optical imaging system being developed by Defence Research and Development Canada (**DRDC**) – Valcartier to enhance the capability of search and rescue (**SAR**) crews to operate effectively at night and in degraded weather conditions. In order to ensure that a SAR operator would be able to use the system effectively and with a minimal amount of training, a prototype human-machine interface (**HMI**) was developed to evaluate design concepts. The latest development phase added sensor controller options and a configurable display interface for evaluating interface design concepts.

Résumé

Le présent manuel de l’utilisateur donne un aperçu des fonctions du logiciel élaboré pour prendre en charge l’examen empirique d’une interface utilisateur simulée d’un système perfectionné de surveillance multi-capteurs intégré (**AIMS**) (anciennement appelé Système perfectionné de surveillance à intensification de lumière visible et à infrarouge ou **ELVISS**). Le système AIMS est un système d’imagerie électro-optique en cours de développement à Recherche et développement pour la défense Canada (**RDDC**) – Valcartier pour améliorer la capacité des équipages de recherche et de sauvetage (**SAR**) à travailler avec efficacité la nuit et par intempérie. Afin de garantir qu’un opérateur du service SAR sera en mesure d’utiliser le système efficacement et moyennant une formation minimale, un prototype d’interface humain-machine (**IHM**) a été développé pour évaluer les principes de conception. La dernière phase de développement consistait à ajouter des options de contrôleur des capteurs et une interface d’affichage configurable pour évaluer les principes de conception d’interface.

This page intentionally left blank.

Executive summary

Introduction

This document provides an overview of software developed to support the empirical investigation of a simulated user interface for an Advanced Integrated Multi-sensor Surveillance (**AIMS**) system (formerly known as the Enhanced Low-Light Level Visible and Infrared Surveillance System – **ELVISS**).

Defence Research and Development Canada (**DRDC**) is developing a multi-sensor system composed of an Active Gated TV (**AGTV**) and a thermal Infrared (**IR**) imaging system. By coordinating the use of a pulsed laser illuminator and AGTV camera, the AGTV component of AIMS provides effective imaging in the absence of ambient light. In addition, the active range gate allows the AGTV system to penetrate meteorological phenomena such as fog, snow and rain much more effectively than a FLIR camera. The FLIR camera is a passive thermal imaging system that produces an image based on temperature variation by detecting mid-infrared and far-infrared radiation. Both sensors are bore sighted and are packaged in a gimballed “ball” that is mounted on the exterior of an aircraft or vehicle. The use of gyros inside the ball allow the camera within to maintain its orientation in the Earth frame of reference, without being affected by roll, pitch and yaw changes in the supporting aircraft or vehicle.

In order to ensure that a SAR operator would be able to use the sensor suite effectively and with a minimal amount of training, a prototype human-machine interface (HMI) was developed to provide a cost effective method for evaluating design concepts and assessing the impact of design characteristics on operator performance. The prototype was developed for the Silicon Graphics, Inc.’s (**SGI**) platform but was eventually ported to run on Microsoft Windows. A series of modifications to the software have enhanced the capability of the research platform. This report document additions in the latest development phases, which include tracking and motion-related functionality, graphical user interface (GUI) display configurability, a larger number of sensors/displays, a thumbnail bar, and a new name, AIMSsim.

Results

In the latest rendition the system, now called AIMSsim, includes the following modifications:

- A capability to over-ride the default joystick controller on the fly panel with a joystick connected via a USB port to allow an evaluation of different types of controllers (e.g., displacement joystick; force feedback joystick)
- A feature that allows the display interface to be completely reconfigured, to accommodate test and evaluation of interface design concepts since the sensor suite now supports five sensors instead of two as previously contained in ELVISS.

- The ability to change the position of the sensor image windows on the interface, relocate the moving map display, and add thumbnails representing each of the sensors. A variety of display options can be modelled.

Significance

The continued development and upgrade of the *AIMSsim* research platform provides the experimenter with an appropriate level of simulation detail to conduct human peformance analyses which in turn delivers up-to-date knowledge and advice on the design of sensor surveillance systems to the military stakeholder. A System Manual (Schoenborn, 2007a) and a Final Report (Schoenborn, 2007b) that summarizes the work performed are also associated with this document.

Schoenborn O., 2007; AIMSsim Feature Development II. DRDC Atlantic CR 2007-300. Defence R&D Canada – Atlantic.

Sommaire

Introduction

Le présent document donne un aperçu du logiciel élaboré pour prendre en charge l’examen empirique d’une interface utilisateur simulée d’un système perfectionné de surveillance multi-capteurs intégré (**AIMS**) (anciennement appelé Système perfectionné de surveillance à intensification de lumière visible et à infrarouge ou **ELVISS**).

Recherche et développement pour la défense Canada (**RDDC**) est en cours de développer un système multi-capteurs composé d’un capteur de télévision commandée par portes actives (**AGTV**) et d’un système d’imagerie thermique infrarouge (**IR**). En coordonnant l’utilisation d’un illuminateur laser et d’une caméra AGTV, l’élément AGTV de l’AIMS assure une imagerie efficace en l’absence de lumière ambiante. De plus, la porte active de distance permet au système AGTV de pénétrer des phénomènes météorologiques tels que le brouillard, la neige et la pluie beaucoup plus efficacement qu’une caméra FLIR. La caméra FLIR est un système d’imagerie thermique passive qui produit une image en fonction de la variation de la température en détectant le rayonnement infrarouge moyen et le rayonnement infrarouge lointain. Les deux capteurs sont à simbleau et sont montés sur cardan dans une « boule » installée à l’extérieur d’un aéronef ou d’un véhicule. L’utilisation de gyros à l’intérieur de la boule permet à la caméra de maintenir son orientation par rapport à la Terre, sans être influencé par les mouvements de roulis, de tangage et de lacet de la plate-forme d’installation.

Afin de garantir qu’un opérateur SAR sera capable d’utiliser l’ensemble de capteurs efficacement et moyennant une formation minimale, un prototype d’interface humain-machine (IHM) a été développé pour fournir une méthode économique d’évaluer les principes de conception et d’évaluer l’incidence des caractéristiques de conception sur le rendement de l’opérateur. Le prototype avait été développé pour la plate-forme de Silicon Graphics Inc. (**SGI**), mais on a fini par le faire migrer vers Microsoft Windows. Une série de modifications au logiciel a amélioré les capacités de la plate-forme de recherche. Le présent rapport décrit les ajouts qui ont été apportés pendant les dernières phases de développement et qui comprennent les fonctions en rapport avec la poursuite et le mouvement, la configurabilité de l’affichage de l'interface graphique (GUI), un plus grand nombre de capteurs/afficheurs, une barre de vignettes et un nouveau nom : AIMSsim.

Résultats

La dernière version du système, appelé maintenant AIMSsim, comprend les fonctions suivantes :

- Capacité de surpasser le contrôleur à manette par défaut sur le boîtier de commande FlyPanel au moyen d’une manette connectée à un port USB pour permettre l’évaluation de différents types de contrôleurs (p. ex. manette de commande de déplacement; manette à retour de force)

- Fonction qui permet la reconfiguration complète de l'interface d'affichage, pour prendre en charge l'essai et l'évaluation des principes de conception, puisque l'ensemble de capteurs compte maintenant cinq capteurs au contraire de celui du système ELVISS, qui n'en comptait que deux.
- Capacité de changer la position des fenêtres d'image de capteur sur l'interface, de déplacer l'affichage cartographique dynamique et d'ajouter des vignettes représentant chacun des capteurs. Possibilité de modéliser une variété d'options d'affichage.

Portée

Le développement et la mise à niveau continus de la plate-forme de recherche *AIMSsim* offrent à l'expérimentateur un niveau de simulation assez détaillé pour mener des analyses du rendement humain, qui fournissent à leur tour à l'intervenant militaire des connaissances à jour et des conseils au sujet de la conception de systèmes de surveillance à capteurs. Un manuel du système (Schoenborn, 2007a) et un rapport final (Schoenborn, 2007b), qui résume le travail effectué, sont également associés au présent document.

Schoenborn O., 2007; AIMSsim Feature Development II. DRDC Atlantic CR 2007-300. Defence R&D Canada – Atlantic.

Contributors

Various	1999 – Apr 2002	CMC Electronics, The HFE Group, DRDC Toronto
Amanda Renner	May – Aug 2002 Jan – April 2003	DRDC Toronto
Lisa Rehak	May – Aug 2003	DRDC Toronto
Alice Malisia	Sept – Dec 2003	DRDC Toronto
Jennifer Jeon	Jan – April 2004	DRDC Toronto
Michael Perlin	May 2004 – Jan 2005	CMC Electronics
Oliver Schoenborn	Feb – Mar 2005	Greenley & Associates
Oliver Schoenborn, Bassem Mikhael	Mar – Oct 2005	Greenley & Associates
Tarra Penney	May 2005 – Apr 2007	DRDC Atlantic
Oliver Schoenborn	Dec 2005 – Aug 2007	CAE Professional Services

Foreword:

This version of the User Manual incorporates all changes to the DRDC AIMS HMI Experimental Prototype, as of June 2007. This document is based on the User Manual originally created by the HFE Group, and the Experimenter's Guide, provided by DRDC. The HFE Group and CMC, early contributors to this document, are no longer responsible for its content. Along with the System Manual, this manual provides a good introduction and reference to the DRDC AIMS HMI Experimental Prototype software, AIMSsim.

Table of contents

List of figures

List of tables

This page intentionally left blank

1 Introduction

1.1 General

Defence Research and Development (DRDC) Valcartier has been developing a flyable prototype of an Advanced Integrated Multi-sensor Surveillance (**AIMS**) system, formerly known as the Enhanced Low-Light-Level Visible and Infrared Surveillance System – **ELVISS**.

In conjunction with the flyable prototype DRDC has been contracting out the development and enhancement of the capabilities for a research platform simulating the human-machine interface (**HMI**) of ELVISS and subsequently AIMS. The research platform allows for the empirical investigation of interface and sensor characteristics on search and detection capability under different simulated environmental conditions.

(For details of the ELVISS interface and operator control panel currently used in the flyable prototype refer to McFadden (2006), and to Baker & Youngston (2006) for a report on interface design concepts for AIMS.)

1.2 Background

DND has identified a requirement to enhance the capabilities of Search And Rescue (**SAR**) operators to conduct operations at night and under degraded weather conditions. To this end, the Defence Research and Development (DRDC) Valcartier is developing a multi-sensor system composed of an Active Imaging System (the Airborne Laser-Based Enhanced Detection and Observation System -- **ALBEDOS**) and a thermal Infrared (**IR**) imaging system. By coordinating the use of a pulsed laser illuminator and AGTV camera, the AGTV component of AIMS provides effective imaging in the absence of ambient light. In addition, the active range gate allows the AGTV system to penetrate meteorological phenomena such as fog, snow and rain much more effectively than a FLIR camera. The FLIR camera is a passive thermal imaging system that produces an image based on temperature variation by detecting mid-infrared and far-infrared radiation. Both sensors are bore sighted and are packaged in a gimballed “ball” that is mounted on the exterior of an aircraft or vehicle. The use of gyros inside the ball allows the camera within to maintain its orientation in the Earth frame of reference, without being affected by roll, pitch and yaw changes in the supporting aircraft or vehicle.

In order to ensure that a SAR operator would be able to use the system effectively and with a minimal amount of training, a prototype HMI was developed to evaluate design concepts. The VAPS HMI prototype (**ELVISS**), developed for the Silicon Graphics, Inc.’s (**SGI**) platform, provided a cost effective method for evaluating the impact of design characteristics of dual sensor systems on operator performance. However the capability was limited and the architecture was not designed to support systematic investigation of the usefulness of the proposed system under different conditions or to manipulate the sensor and interface characteristics.

The ELVISS VAPS prototype was therefore extensively enhanced to allow the empirical investigation of different interface and sensor characteristics on search and detection capability under different environmental conditions. Included in this upgrade was a Scenario Generation Environment (SGE) that provided user-friendly capability for generating scenarios. Nonetheless, despite the increased versatility of the prototype the requirements of a specific experimental design required that Lua scripting be used to make additional modifications to the software. While further development drastically expanded the Lua scripting capabilities, the SGE was not similarly extended and some of its scenario-generation capabilities became incompatible with the prototype. Thus Lua scripting in now the primary mode of control of the prototype.

The prototype HMI was then ported to run on Microsoft Windows. This involved replacing the VAPS and SGI Performer™ with equivalent functionality using the OpenSceneGraph open source graphics library. The robustness and traceability of the system were significantly improved. The latest development phases added (amongst other things) important tracking and motion-related functionality, GUI display configurability, a larger number of EO and IR sensors/displays, a thumbnail bar. Since 2006, the software system is known as AIMSsim.

1.3 Aim

The aim of this report is to provide instructions for installation and use of software developed to support empirical investigation of a simulated user interface for the AIMS Human Machine Interface (HMI).

1.4 Objectives

The specific objectives of this report are to:

a. Describe the process of installing AIMSsim;

b. Provide instructions on the use of AIMSsim software

1.5 Report Outline

The report is structured as follows:

1. Section One describes the background, aim and scope of this document;
2. Section Two explains the process of installing AIMSsim;
3. Section Three documents the experiment development and execution process (**EDEP**) to follow when using AIMSsim;
4. Section Four documents the use of the main component of AIMSsim; and
5. Various annexes document the scripting functions and predefined variables, predefined system events and messages, target object types, OpenFlight file conversion to other formats via OpenSceneGraph, and the AIMS Scenario Generation Environment (SGE).

2 Installation

2.1 General

The following subsections describe the process of installing the AIMSsim components and supporting software. The instructions in this section assume a basic working knowledge of the Microsoft Windows operating system and understanding of the AIMSsim System Manual, especially with regards to *simControl* and *simDisplay*. The instructions also assume that you have logged onto the system and have read and write privileges for the target directory into which you intend to install the AIMSsim software.

2.2 Before You Begin

Before you begin installing AIMSsim:

- Ensure that the system requirements defined in AIMSsim System Manual have been met.
- Ensure that you have adequate permissions to write data to the target directories into which you intend to install the software.
- Ensure that you have the required amount of hard-disk space before installing, i.e. approximately 1G, due to the large size of the included terrain databases.

2.3 Installing the AIMSsim Software: First Time

1. Locate the compressed archive containing the software. It should be named *AIMSsim-*-bin.tar.gz*, with * being a number of the form 1_2_3 indicating major version 1, minor release 2, sub-minor/patch release 3.

2. Extract the contents of this file to a location of your choice. This will create a folder, which we call **AIMS_HOME**, containing:
   - the binaries for the HMI prototype software with supporting DLLs,
   - the SGE executable,
   - subfolders for some experimentation scripts,
   - and a copy of the System and User manuals.

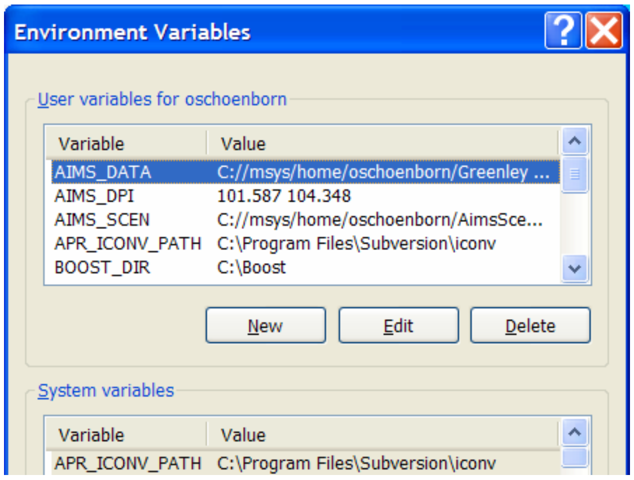
3. (Optional) Create a shortcut to the SGE executable on your desktop.

4. Extract the contents of the AIMSdb_*.zip file you obtained to a location of your choice. This is the visualization database (referred to in this document as AIMS_DB).

5. Connect the FlyPanel to the computer, according to the FlyPanel manual. This basically consists in connecting one FlyPanel cable to one of your PC's serial ports (the same as specified by FB_PORT below), and the FlyPanel's power cable to a power outlet.

6. (Optional) Connect a USB joystick to an available USB port if you want to override the FlyPanel's joystick with the USB joystick.

7. Define the following AIMSsim environment variables (see Figure 1):
   a. AIMS_DATA (required): points to the AIMS_DB folder.
   b. AIMS_DPI (required): a space-separated pair of values that define the horizontal and vertical dots-per-inch of your monitor. This is necessary for an accurate scaling of the Moving Map Display. E.g. a 21" monitor has a screen 15.75" x 11.5". At a resolution of 1600 by 1200, this would imply an AIMS_DPI equal to "101.587 104.348".
   c. AIMS_SCEN (required only if you want to use the SGE): points to a folder of your choice, where the SGE will save and load project files. Note that you can easily override this from the SGE through its File Browser.
   d. FB_PORT (required only if default inadequate): defaults to "COM1"; set it to some other appropriate value (e.g. "COM2" or "COM3") depending on which serial port your FlyPanel is connected to.
   e. AIMS_ATD_CUES (optional): enable/disable Automatic Target Detection (ATD) cues functionality.



Variable	Value
AIMS_DATA	C://msys/home/oschoenborn/Greenley ...
AIMS_DPI	101.587 104.348
AIMS_SCEN	C://msys/home/oschoenborn/AimsSce...
APR_ICONV_PATH	C:\Program Files\Subversion\iconv
BOOST_DIR	C:\Boost

Variable	Value
APR_ICONV_PATH	C:\Program Files\Subversion\iconv

Figure 1: AIMS environment variables defined

2.4 Upgrading the AIMSsim Software

Upgrading the application is necessary when a new version of the software is released by CAE Professional Services. The release will be a file named *AIMSsim-*-bin.tar.gz* (as described in section 2.3) but the * will be a different version number from the file downloaded the first time. Moreover, some of the steps from the previous section are not necessary.

CAE PS recommends these steps to ensure robust operation of your system:

1. Locate the folder containing your AIMS_HOME (see section 2.3). E.g. if your experiments are in *c:\AIMSsim\AIMSsim-2_2_3*, then the folder is *c:\AIMSsim*
2. Extract the compressed archive that you received to this folder. This will create the same content as step 2 of section 2.3. E.g. if the new compressed archive is *AIMSsim -2_3_1-bin.tar.gz*, and the recipient folder is *c:\AIMSsim*, then a new folder *c:\AIMSsim\AIMSsim-2_3_1* will have been created and represent your new AIMS_HOME.
3. (Optional) Make the SGE shortcut point to the new SGE.
4. Copy your scenarios from previous AIMS_HOME (e.g. *c:\AIMSsim\AIMSsim-2_2_3*) to your new AIMS_HOME (e.g. *c:\AIMSsim\AIMSsim-2_3_1*)

3 Experiment Development and Execution Process

The instructions in this section assume a basic understanding of the AIMSsim System Manual, especially with regards to *simControl* and *simDisplay*. This section also requires a basic understanding of scripting. Knowledge of Lua scripting is not necessary but will be useful.

AIMSsim represents virtual prototypes of proposed AIMS system interfaces. A virtual interface allows you to test and evaluate the impact of different interface and sensor characteristics on search and detection tasks by simulating, in real time, “experiments” (also known as “scenarios”) described by scenario text files, and by allowing you to collect data on how well the system was operated in accomplishing the goals set out by the scenario.

Usage of AIMSsim involves several steps:

1. Develop experiment:
    a. Design your experiment (conceptually)
    b. Convert it to a Finite State Machine (FSM), diagrammatically
    c. Determine which scripts are available for re-use by looking at previous experiments, and the LuaUtils folder in AIMS_HOME
    d. Create a directory, in AIMS_HOME, for your new experiment
    e. Create the Lua initialization script for your experiment, or copy it from another experiment and edit it
    f. Create the remaining Lua scripts for the FSM and other tasks (designation, etc), or copy from existing ones and edit
    g. Run *simControl* to test and debug the experiment; this involves looking at the visual display for proper sequence of screens, and proper changes of states and settings; as well as looking for any messages in the DOS shell or in the log files *sim*_log.txt* that get created by each process in AIMS_HOME, and looking at any collected data for correctness
    h. Edit the experiment’s Lua scripts until experiment works as required
2. Execute experiment:
    a. Run *simControl* to execute the experiment and collect data
    b. Analyze data

Steps 1d to 1f result in a set of Lua scripts, referred to as your Experiment Simulation (ExS) Scripts.

In older generations of AIMSsim, steps 1d to 1f did not involve Lua scripts but plain text files with value settings. These value files could be created from the AIMS *Scenario Generation Environment* (SGE) and read by AIMSsim. Since the scripting capabilities were added to AIMSsim, only the flight plan and the targets files created by the SGE can be read into *simControl*. Settings saved by the SGE in the SGE project configuration file (such as environmental conditions etc -- see the AIMSsim System Manual) can no longer be read by *simControl*. Due to the loose coupling between the *simControl* and the SGE, use of the SGE is described in detail in Annex H:.

3.1 Experiment Simulation (ExS) scripts

Lua scripting was added to the AIMS HMI Prototype as the means of describing experiments. Lua is a relatively simple language that is interpreted at run time and uses a C like syntax. For official information about Lua, visit www.lua.org.

While Lua is easy to use, it is at the same time a very powerful language. One major departure from C and many other languages is that variables do not have to be declared before they are used. This eases development, but makes debugging a little more difficult. Be sure that you are spelling variables correctly. Also, it is suggested that each script contain a comment in the first few lines of the script, describing all global variables that are assumed to exist when the script is called.

One Lua script is required to initialize the AIMSsim system, and one Lua script is possible per state transition in the FSM.

All standard Lua library functions are available for use in your experiment scripts. In addition to these standard functions, there are a large number of AIMSsim functions and settings that have been made available through the scripting interface. These are the functions that will be the most useful for implementing experiments. See Annex A: for a table of all exported AIMSsim functions available to your scripts.

3.1.1 Available experimentation facilities

There are several AIMSsim facilities that have been exposed to aid in experiment implementation:

- **User Timer**: a timer exists to allow response times to be captured through scripts. It is up to the experimenter to ensure that this timer is reset and read at the appropriate time, via the *ResetUserTimer*() and *GetUserElapsedTime*() functions. Note that Lua provides timing functions natively, however they provide only 1 second resolution.

- **Logging**: Text messages are generated by the system throughout its execution to help the experimenter identify and correct problems. All text messages are time stamped, and are output via one of several *loggers*, described in Table 10. Each logger has a name and a different purpose, e.g. the “info” loggers output messages of a purely “informative” nature, whereas the “err” loggers output error messages. Each logger’s state (enabled/disabled) and outputs can be changed in *simControl*’s and *simDisplay*’s scripts. The system also allows the experimenter to output information from the Lua scripts via the *Log()* function, which is associated with the “user_log” logger. The big advantage of loggers is uniformity of output, time stamping of all messages, and the ability to filter the output based on logger. More details are given in section 4.2.14.

- **Experiment execution trace**: every call to exported functions and every change to exported settings can be logged so as to get a “trace” (in programmer parlance) of the script execution. Each trace message says what has been done, what new value has been set etc. Very useful while developing an experiment.

- **User Variables (deprecated)**: they exist in 3 forms, integers, integer arrays, and float arrays. User variables are uniquely identified by their names, and are created as the user sets them. They were originally provided as a safe way to share data between scripts, as they differ from native Lua global variables in two ways:
    - How they are accessed: user variables are accessed via the *Set*()* and *Get*()* AIMSsim functions as quoted arguments, whereas Lua variables appear directly Lua script expressions, unquoted.

```
-- Initialization script
setInt(“hello”, 3);       # User variable named “hello”
-- Some other script
hello = GetInt(“hello”);  # Get its value, put in Lua variable
goodbye = hello + 2;      # change Lua variable
setInt(“hello”, goodbye); # commit change to User variable
```

    - Behavior when undefined: If a script tries to access an undefined user

```
local a = 1
local b = a+2
print(b)
```

      variable, it will get 0, whereas an undefined Lua variable is nil and will cause an error message to be printed to the shell window.

This feature has been deprecated because it bypasses one of the fundamental validation mechanisms available in Lua: exceptions. If a script run by *simControl* attempts to use a Lua native variable that is undefined, simControl’s Lua interpreter will raise an exception and log an error message to the *simControl_log.txt* log file (see “Logging”, above). An undefined user variable appears as a zero in your scripts, which will rarely cause any errors, let alone a meaningful error message identifying file and line where the error occurred; if you are lucky, you may eventually notice unexpected behavior, but this will often not be the case, and is much more difficult to troubleshoot.

Lua variables are by default global. Variables declared in one script are available in other scripts run during the experiment (because a single Lua virtual machine is used). The use of the “local” qualifier on variables is encouraged as a mechanism to minimize the number of global variables that exist during an experiment:

This helps decrease the likelihood that a variable meant only for local use in a script has the same name as one of your “undefined” global variables.

3.1.2 Typical initialization tasks

Some typical tasks done in an initialization script:

- Set up AIMSsim configuration variables (FOV, map mode, etc)
- Set up some global run-time constants used by your experiment
- Define terrain database to use

- Define path plans
- Define targets and each one’s attributes
- Add path plans to targets and/or aircraft
- Define FSM
- Define run-timed and flight-timed transition events (e.g., to turn off the MMD at specific times of the run or of the flight)
- Define scripts to be run at every time step (so-called “periodic” scripts)
- Schedule an event to transition away from the “INIT” state after initialization script has completed

The following box shows a relatively advanced example of AIMSsim initialization Lua script, performing several of the above tasks. Refer to section 4 to gain a better understanding of the above tasks.

```
Trace(1)

function log(logFile, msg)
    local curout = io.output()
    io.output(logFile)
    io.write(msg, "\n")
    io.output(curout)
end

dofile("Designation/initWorld.lua")
dofile("Designation/initFSM.lua")

-- use the look-at algorithm for autotracking
autoTrackByLookat = true
autoTrackDesignatedTarget = true
userResponse2Change = {} -- empty table to store results


-- Info for saving field of view measures
fovScript = "Designation/outFOV.lua"
fovFName  = "Designation/results/fov.txt"
fovFile   = io.open(fovFName, "w")
if fovFile then
    AddPeriodicScript(fovScript)
else
    print("Could not open '"..fovFName.."', can't save FOV")
end

-- Open a file to save tracking measures; will use stdout if for some
-- reason file can't be opened.

trackingFName = "Designation/results/tracking.txt"
trackingFile = io.open(trackingFName,"w")
if not trackingFile then
   print("Can't open file '"..trackingFName.."', can't save tracking measures, using stdout
instead")
end

-- Define the "save" function to save tracking measurements, needed by tracking script
function saveTrackingMeasure(targetDistance, targetRelSpeed, targetLOSH, targetLOSP)
    msg = string.format("%.2f %.2f %.2f %.2f %.2f\n", GetRuntime(),
           targetDistance, targetRelSpeed, targetLOSH, targetLOSP)
    if trackingFile then
        trackingFile:write(msg)
    else
        print(msg)
    end
end

-- and the comment function for outputting comments to file
function commentTrackingMeasure(comment)
    if trackingFile then
        trackingFile:write(comment)
    end
end


-- We're done, notify system that we can switch to next state
GenerateEvent("EVENT:initDone")
```

Figure 2. Example AIMSsim ExS initialization script

3.2 Programmable Finite State Machine

The flow of an experiment can described using a finite state machine (FSM). In order to provide the capability to run the largest number of possible experiments, the scriptable architecture allows the state machine that describes an experiment to be set at initialization time through scripts. This allows for any ordering of screens and behavior to be created. AIMSsim should be able to behave according to any desired experimental flow using the facilities provided. It should be noted, however, that care must be taken in describing the FSM of an experiment, since there is no facility to ensure that the FSM described makes sense.

A FSM can be viewed as a graph with the nodes representing states, and the arcs representing state transitions (see the System Manual). By describing all the transitions, the FSM can be fully described. Transitions are described using four fields: the current state, the next state, the triggering event, and the action to take. This is done via the exported *AddTransition*() function, normally during the initialization script. Only one transition <state, event> pair is possible.

The states can be named as desired by the experimenter with the following two exceptions: the initial state is always called "STATE:INIT", and the final state is always called "STATE:EXIT". In addition, all states must start with the name "STATE:". It is up to the experimenter to provide a transition from the "INIT" state. A good way of doing this is by having the last line of the initialization script call the *GenerateEvent*() function with the proper event name defined in the experiment's FSM. It is also up to the experimenter to provide for a transition into the "EXIT" state that will cause the system to exit.

State transitions within your experiment's FSM are executed in response to *events*. Each transition causes a corresponding Lua script to be run, if one was specified in the initialization script. There are a number of ways for events to be triggered:

1. System events are automatically triggered by the system under specific conditions. These events have set names and are described in Annex C:. E.g., when the exit screen is shown with the exit button, an EVENT:EXIT_PRESSED event is generated when the user presses the button.
2. Timed events can be added using the *AddTimedEvent*() and *AddFlightTimedEvent*() functions. Timed event names are chosen by the experimenter. A timed event occurs at a specific runtime or flight time, and causes an event to be generated.
3. Generic events can be generated within scripts using the *GenerateEvent*() function. Generic events can be named as desired by the experimenter, except for the constraint that all events must start with the word "EVENT:".

The following table shows an example experiment, and associated states and events. System State names and System Event names are in italics, all others are arbitrary names and specific to this simplified example:

Table 1. Example AIMSsim experiment

	Experiment stage	**State Name**	**Lua script to execute on entry to state**	**Transition event**
1.	Initialize	*STATE:INIT*	initialization script	EVENT:DONE_INIT
2.	Show the startup screen, wait for setup complete: operator ready, database loaded, etc.	STATE:SHOW_START	toStartup.lua	Press “Start” (EVENT:*START_PRESSED*)
3.	Show operator interface, but no operation allowed; the aircraft flies along prescribed flight path; wait for timed event or end of flight path	STATE:FLYING	toFlying.lua	Timed event (EVENT:SEARCH_TIME) or flight path done (*EVENT:END_OF_FP*)
4.	At predefined points in time, freeze aircraft, hide map, add a target to landscape, and allow operator to use joystick controls to move sensor cameras until target found; wait for them to press "button 9" on their control	STATE:SEARCHING	toSearching.lua	Button 9 clicked (*EVENT:B9_PRESSED*)
5.	Take a snapshot of current camera view and save it	STATE:CAPTURE_IMAGE	toCapture.lua	Image saved (*EVENT:IMAGE_CAPTURED*)
6.	Resume flight path and reactivate map; ie aircraft starts moving again, but operator can't control anything; prepare for next target with timed event, **OR (item #8)**	STATE:FLYING	toFlying.lua	Timed event (EVENT:SEARCH_TIME)
7.	No more targets: continue flying until flight path completed			Flight path completed (*EVENT:END_OF_FP*)
8.	Flight path completed. Save data, close files, etc.	STATE:EXITING	toExit.lua	
9.	Exit program	*STATE:EXIT*		

The corresponding finite state diagram would be as shown in Figure 3.

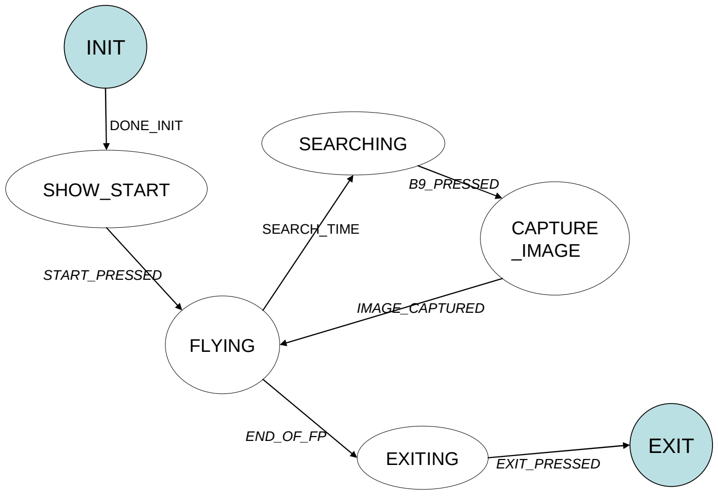


Figure 3. Possible FSM for example experiment

The Lua script `stateMachine.lua`, executed by the example initialization script in

Figure 2, is shown in Figure 4. This shows how the recording of a specific response by the user (LOC#_PRESSED), one of 8 choices, uses a transition script specific to the response. It is up to the script to make sure that the system returns to the FLYING state (e.g., by itself calling `dofile(“toFlying.lua”)`; however the FSM could be modified to use an event instead).

```
AddTransition("STATE:INIT", "STATE:STARTUP",
       "EVENT:DONE_INIT",    "./SC/generic/toStartup.lua")
AddTransition("STATE:STARTUP"," STATE:FLYING",
       “EVENT:START_PRESSED", "./SC/generic/toFlying.lua")
AddTransition("STATE:FLYING"," STATE:EXITS",
       "EVENT:END_OF_FP",    "./SC/generic/toExit.lua")
AddTransition("STATE:FLYING"," STATE:SEARCH",
       “EVENT:SEARCH_TIME", "./SC/generic/toSearch.lua")
AddTransition("STATE:SEARCH"," STATE:CAPTURE",
       "EVENT:B9_PRESSED",   "./SC/generic/toCapture.lua")
AddTransition("STATE:CAPTURE","FLYING",
       "EVENT:IMAGE_CAPTURED", "./SC/generic/toQuestion.lua")
AddTransition("STATE:EXITS","STATE:EXIT","EVENT:EXIT_PRESSED", "")
```

Figure 4. Example FSM

Note however that for a non-trivial experiment, there can be many different FSM representations. Your choice should be based on clarity and modularity, rather than brevity or ease. The extra effort will pay off when debugging and extending your experiment.

The following list outlines *some* of the tasks that could take place in the above example.

- toStartup.lua:
    - tell *simDisplay* to show startup screen
    - create “target event number” to count how many targets have been found
- toFlying.lua:
    - reset user time to 0 every time entering this state (as if each waypoint was the beginning of the simulation)
    - tell *simDisplay* to update the display, because targets are removed in other scripts (like toCapture.lua)
    - disable joystick input and resume the aircraft motion along flight path
    - set sensor heading since defaults to 0 pitch, or if not first time run, operator has likely changed it while search for target
    - start updating sensor heading periodically so it follows aircraft direction
    - increase "target event" number
    - tell *simDisplay* to switch to "operational" screen
- toSearch.lua:
    - clear the targets set (removes all targets from display)
    - stop all ongoing data collection
    - enable joystick input
    - pause flight path
    - play sound (e.g. to alert operator)
    - get desired heading and pitch of target, using target and bearing table
    - add a target there and tell *simDisplay* to update its scene graph
    - reset user timer, to time how long it takes operator to find the target that was just placed
    - tell AIMSsim to run streamLog.lua every time step during search, to capture sensor positions to file, e.g. for debugging
- toCapture.lua:
    - Button 9 has been pressed, so operator has made decision: stop script and clear target(s)
    - get time take for search/find
    - tell *simDisplay* to capture current camera view (of which sensor?); once captured, *simDisplay* generates IMAGE_CAPTURED event automatically
- toQuestion.lua:
    - camera view has been capture so reset user timer to time how long taken to answer upcoming question
    - get the question type for current "target event" number
    - tell *simDisplay* to display the question string
- toExit.lua:
    - tell *simDisplay* to exit too

4 Common Experiment Tasks

4.1 Starting and Exiting the AIMSsim HMI Prototype

AIMSsim must be started at a DOS command line (though a desktop shortcut could be created). Command line parameters are options that are provided to *simControl* to specify some of its run time parameters. In order for the HMI Prototype to function correctly, one command line parameter is required for the initialization script to use, and a number of optional parameters are possible. The basic steps for running the HMI Prototype:

- Open a command shell, e.g. by doing *Start->Run…->"cmd"*
- Cd to the folder containing the AIMSsim binaries (AIMS_HOME)
- Run

  ```
  simControl -i initscript.lua -p subject_number -t trial_number
  ```

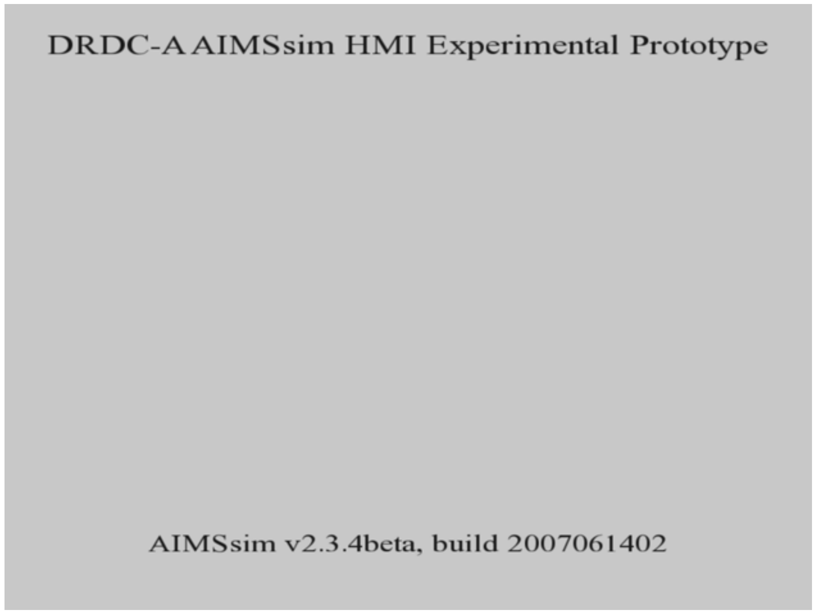
  Where:
  - *initscript.lua* is the filename of the AIMSsim Lua initialization script, relative to the current directory; e.g. TEST/main.lua. Note that path must be formatted in universal form rather than DOS form, i.e. use forward slashes as folder separators.
  - *subject_number* and *trial_number* are optional numeric parameters used to identify and name the data collection files that will be created by AIMSsim. If *subject_number* and *trial_number* are not provided, AIMSsim will assign default values of **0** and **0**, respectively.

Most environment variables that were available in previous generations of AIMSsim have been removed or replaced with corresponding AIMSsim variables exported to the the Lua scripting engine. Refer to AIMSsim System Manual for a list of all environment variables currently supported, and to section 2.3 of the current manual for example settings of some of them.

AIMSsim will give best results when it is run covering the entire screen of the monitor. A useful key combination is ALT-tab, which cycles between open windows of the desktop.

4.1.1 Initial View

Once started, *simControl* spawns off two other processes, namely *simInputs* and *simDisplay*. The former is to read the controls box (e.g. a FlyBox™), the latter is the display side of the system, as described in the *AIMSsim System Manual*. *SimControl* then enters the "INIT" state, and loops forever, waiting for events. It is up to the initialization script to transition *simControl* out of the "INIT" state, but it is up to one of the FSM scripts to transition it to the "EXIT" state according to the desired flow of the experiment. *SimDisplay* will show an essentially empty screen (see Figure 5).


DRDC-A AIMSsim HMI Experimental Prototype

AIMSsim v2.3.4beta, build 2007061402

Figure 5. AIMSsim display in "INIT" state

The capabilities of AIMSsim (i.e. *simControl* and *simDisplay* together) between the "INIT" and "EXIT" states are entirely determined by the AIMSsim and Lua functions and variables used in the Lua scripts.

4.1.2 Controls

There are two types of controls available to the experiment: analog and digital. The analog controls are continuous variables that vary between -1 and 1. The association between a given analog control and its effect in AIMSsim is hardcoded, though in some cases it can be disabled:

- There is a joystick to rotate the camera's view. This is operational only when the exported *orientationControl* variable is `ORIENT_CTL_OPERATOR`.
- There is a zoom in/zoom out function that uses the two FlyPanel levers for continuous zoom: the left lever controls the EO sensors zoom and the right lever controls the IR sensors zoom.

All digital controls are either on or off and generate an event when pressed or released. It is up to the experiment designer to decide how to make use of each (or any) digital control in their transition graph (FSM). All digital control event names have the form `EVENT:`*`CONTROL`*`_PRESSED` and `EVENT:`*`CONTROL`*`_RELEASED`, where *CONTROL* is described in Table 7.

Note that if a USB joystick is also connected to the PC when the application starts, *simInputs* will connect to it and attempt to override the joystick of the FlyPanel with the state of the USB joystick. Other controls on the FlyPanel (such as the 2x4 grid of buttons and the slide levers with buttons) are not overridden and remain available. The event names are not affected by the presence of the USB joystick; a connected joystick merely causes some of the events generated by simInputs for simControl to stem from USB joystick handling.

It is likely that some USB joysticks will have to be configured in order to operate as expected. Most modern USB joysticks, such as Logitech's Extreme 3D, provide a configuration window or application allowing the user to remap the control indices. Use the Table 9 and Figure 17, both found in Annex D, to set the proper mapping of joystick control overrides.

E.g., Table 9 shows that button #1 is always interpreted as the main joystick trigger, i.e. it will cause `EVENT:JOYSTICK_TRIG1` event to be generated when pressed. The joystick configuration application will allow you associate a different button control (physical button on the joystick) to the button known to it as "button #1".

4.1.3 Configuring the Operator View

Figure 6 shows the default operator view configuration. This screen can be reconfigured at startup time via functions in the `Sim.OpsDisplay` module. These are summarized in Annex B.

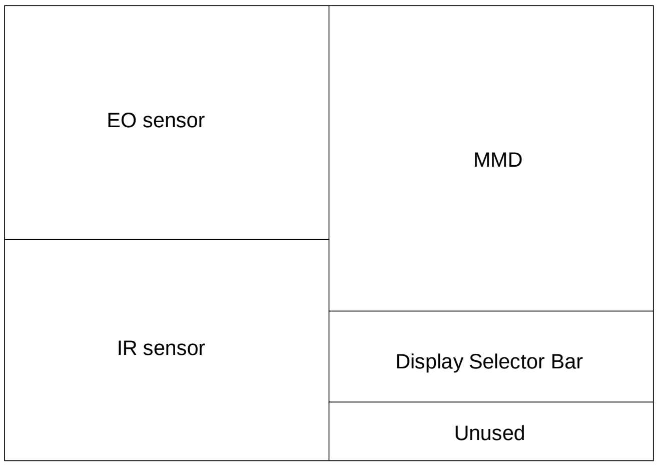


Figure 6. Default Operator View configuration

The "EO sensor" viewport shows only EO type sensor outputs, whereas the "IR sensor" viewport show only IR type sensor outputs. The display selector bar contains 5 thumbnails: AGTV, EO_N, EO_W, FLIR, THIR. The first three are EO, the last two are IR.

The Operator View screen consists of a hierarchy of sub-displays. Each sub-display has a name, consisting of an "individual" name and the "parent" names chained together. Table 2 shows the list of individual names that have special meaning to AIMSsim.

Table 2. Special sub-display individual names

ROOT	root display, occupies the whole Operator View screen
MMD	the sub-display where the MMD will be shown
MAIN_SENSORS	the sub-display where EO sensor displays will be output
AUX_SENSORS	the sub-display where IR sensor displays will be output
SELECTOR_BAR	the sub-display where the Display Selector Bar will be shown
POLAR_PLOT	the sub-display where the aircraft polar plot will be shown
UNUSED_DLG	the sub-display where the “unused dialog area” will be shown

A sub-display’s individual name is used to determine what, if anything, should be displayed in it. If the individual name is one of the above special names, the sub-display is used for the corresponding type of output. Otherwise, it is assumed to contain children sub-displays. Also, if no sub-display is created with one of the above special names, the corresponding component will not be visible in the Operator View.

Figure 7 shows an example of hierarchical subdivision of the Operator View, into the following tree:

```
ROOT
   |-- ROOT.left
   |      |-- ROOT.left.MAIN_SENSORS
   |      \-- ROOT.right.AUX_SENSORS
   \-- ROOT.right
          |-- ROOT.right.MMD
          |-- ROOT.right.SELECTOR_BAR
          \-- ROOT.right.UNUSED_DLG
```

If no display had been named “MMD”, the MMD would not be visible (and its space would be occupied by something else, such as the unused dialog area).

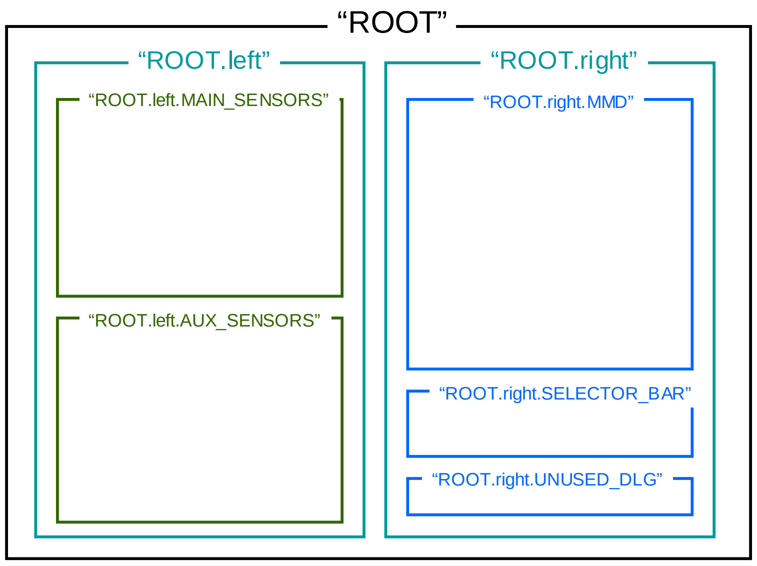


Figure 7. Example of hierarchical subdivision of Operator View

The subdivisions are constrained to take place either along X or Y, but not both, via the exported functions available in the `Sim.OpsDisplay` module:

```
DivideX(parentName, child1Name, fraction1, child2Name, fraction2, …)
DivideY(parentName, child1Name, fraction1, child2Name, fraction2, …)
```

The parent name is the full name of the parent within the hierarchy. The first child's individual name will be *child1Name*, but its full name will be *parentName.child1Name* (note the dot between the two elements of the name).

Each child is given a "fraction" between 0 and 1 of the parent sub-display to occupy, though the final fraction can be omitted if it is 0. Naturally, all fractions must be in the range 0 to 1, and the total of all the fractions must not exceed 1. As a convenience to the experimenter, any children sub-displays that have *fraction*=0 will share the unalloted space of the parent. E.g.

```
Sim.OpsDisplay.DivideX(
      "ROOT.left", "top", 0.5, "center", 0, "bottom")
```

Assigns fractions of 0 to `ROOT.center` and `ROOT.bottom` therefore they will each occupy 0.25 of the `ROOT.left` display.

One more sub-display function is available:

```
NewChild(parent, childName, x0, y0, fractionX, fractionY)
```

which permits the creation of a sub-display by "attaching" it to its parent rather than by sub-dividing the parent. The x0, y0 are in range [-1,1] and identify the point to attach to, on the parent, while the fractions are the size relative to the parent size. This can be used, e.g. to overlay the polar plot atop other displays. The following example will attach the polar plot to the bottom center of the "main sensors" viewport, and make it 0.15 of the horizontal and vertical sizes of that viewport:

```
Sim.OpsDisplay.NewChild(
      "root.left.MAIN_SENSORS", -- parent
      "POLAR_PLOT",             -- child name
      0, -1,                    -- x0, y0
      0.15, 0.15)               -- size X, size Y
```

Note that the viewports are drawn in the following order:

```
MMD
UNUSED_DLG
MAIN_SENSORS
AUX_SENSORS
SELECTOR_BAR
POLAR_PLOT
```

4.1.4 Troubleshooting startup problems

It is common, while creating new experiment scripts or while doing significant modifications to existing scripts, to see the AIMSsim software “immediately” exit before it reaches the end of its initialization phase, and before any graphics appears. The software typically has time to log an error message (it is extremely rare for it to crash), and exits gracefully. Premature exit could happen due to a syntax error in one of the Lua scripts, or due to an invalid function call from one of the scripts processed during initialization.

Since the logs are not printed to the console by default, it can be confusing at first. Several techniques can be applied to help with figuring out what the problem is:

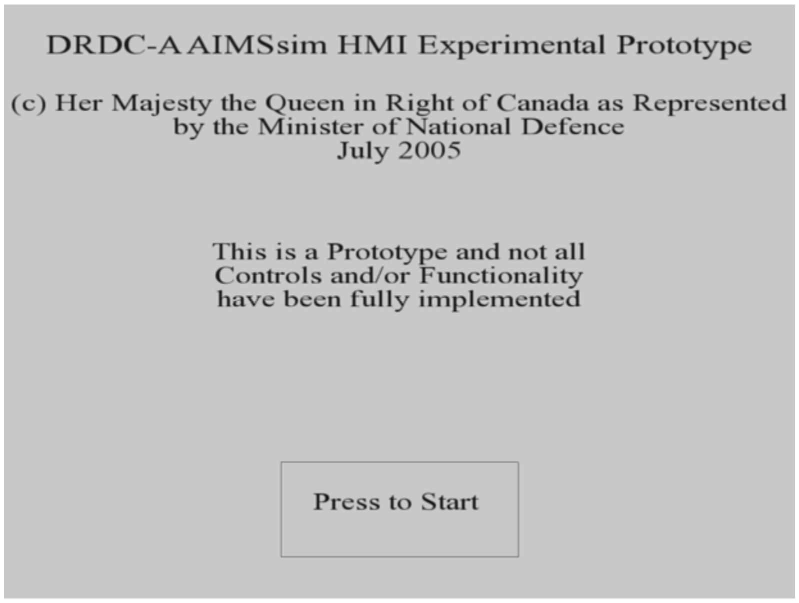
- Duplicate the log output to the console by doing `LogAddOutput(“stdout”)` near the beginning of simControl’s initialization script. Then rerun the software. This time the log output will appear on the screen (as well as in the log file).
- Alternately, simply open the *simControl_log.txt* file and scroll to the end, to see what error message it printed just before exiting.
- See if the system reaches the end of the INIT state by looking for an info line that says “Beginning FSM Loop”. If it is there, the initialization was successful and the error is most likely due to a periodic script.

4.2 Common experiment tasks

Some tasks or activities that you will likely perform in your Lua scripts may not be obvious from the exported AIMSsim functions and variables shown in Annex A:. A few of them are described here. Also, the available screens are documented here. A good way of acquiring a good understanding of how to use AIMSsim is by looking at some of the sample scripts in the `TEST`, `LuaUtils` and `Designation` folders.

4.2.1 Wait to start

If you want the user to have to press “Start” before some stage of the experiment, a “Press Start” screen is available, shown in Figure 8. This can be used to start the experiment after everything is loaded and initialized, or between two phases of the experiment, etc. Simply send a message to *simDisplay* with `SendMessage(“toStartup”)`. When the operator clicks “Start”, the `EVENT:START_PRESSED` event is generated.



DRDC-A AIMSsim HMI Experimental Prototype

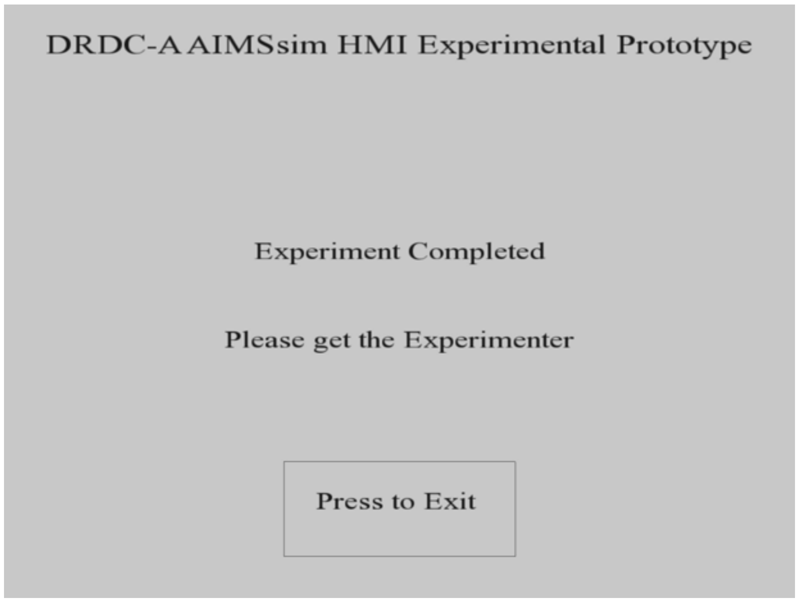
(c) Her Majesty the Queen in Right of Canada as Represented by the Minister of National Defence
July 2005

This is a Prototype and not all Controls and/or Functionality have been fully implemented

Figure 8. Available AIMSsim "Press Start" screen

4.2.2 Wait to exit

An exit screen waiting for the user to press the “Exit” button is available, see Figure 9. This can help the operator know that the programming is terminating normally. When the operator presses “Exit”, the `EVENT:EXIT_PRESSED` system event is generated.


DRDC-A AIMSsim HMI Experimental Prototype

Experiment Completed

Please get the Experimenter

Figure 9. Available AIMSsim "Exit" screen

4.2.3 Show Operator screen

The screen that shows the sensor camera views as well as the map view is called the “Operator” screen. Which views are active, as well as various sensor and map characteristics which will affect how things are displayed, can be set via the many exported AIMSsim variables shown in Annex A:.

This screen is activated by a *SendMessage*(“toOperational”). This screen typically involves a substantial amount of visual detail, such that activating it may take a long time (many seconds). During this time, the simControl is still running, but simDisplay doesn’t respond or update the screen. This can be somewhat confusing to the user but it is up to the experimenter to warn them.

Whenever the Operator screen becomes visible, a `EVENT:OPERATIONAL_UPDATED` event is generated, for the benefit of your ExS scripts. This could be useful, e.g., to have the aircraft start its motion on its flight path only once the Operator screen is visible.

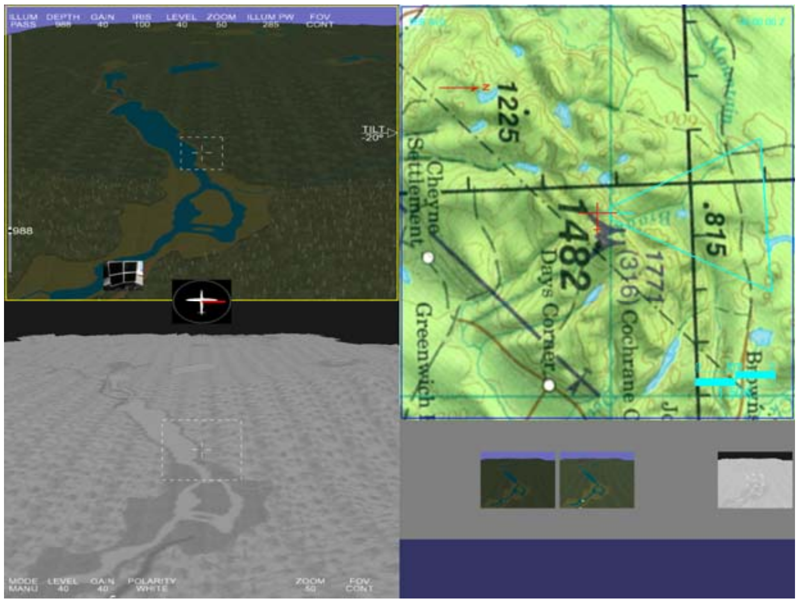


Figure 10. Available AIMSsim "Operator" screen

4.2.4 Switch visible sensor displays in Operator Screen

The Operator Screen, or Operational View, allows only 2 out of 5 sensor outputs to be visible at a time. The two “sensor viewports” can be sized and positioned using the simDisplay configuration file named displayConfig.lua, but by default the EO viewport is in the top left corner of the screen, whereas the IR viewport is in the bottom left corner.

To view a different sensor display, click on one of the display thumbnails in the “Display Selector Bar”. The thumbnail will become invisible, and its associated sensor output will become visible in the EO or IR viewport, depending on whether the sensor is an EO or IR type sensor. Simultaneously, the previous sensor output, being no longer visible, will become visible as a thumbnail. The order of thumbnails is always the same and cannot be configured.

4.2.5 Camera motion or tracking

The default behavior of the system is to mimic the behavior of the camera gimbal of the real system, which is to leave the camera “free” to maintain its orientation in space regardless of the aircraft orientation, and respond to operator input. This is the `ORIENT_CTL_OPERATOR` mode of the setting *orientationControl*.

If the script should take over the camera orientation, e.g. for “ground track mode” (autotracking), the orientation control must be changed to ORIENT_CTL_CPU. Only then your scripts can call *SetSensorOrientation*(), typically via a periodic script added with `AddPeriodicScript`(“scriptname”).

The *SetSensorOrientation*() function specifies the sensor orientation via heading and pitch angles. When autotracking a faraway target while zoomed in, small numerical errors in the view matrix created from the heading and pitch can lead to visible amount of “jitter” for the target being tracked. This is realistic since in a real system, the gimbal motors would be controlled similarly. However, should you desire to remove any form of jitter, a third orientation control mode is available: `ORIENT_CTL_LOOKAT`. This generates the view matrix by telling simDisplay which point in space to look at (via the *SetDisplayLookAtPoint*() function), almost completely removing any jitter.

4.2.6 LOS and Isect

The Line-Of-Sight information is computed by *simDisplay* at every time step and is made accessible to the ExS scripts via *GetLOSPosition*(). Because the scripts are run from *simControl*, which runs asynchronously in parallel with *simDisplay*, *GetLOSPosition*() gives the latest LOS accessible to *simControl*. The *simDisplay* may have in the meantime changed the view. Note however that this kind of lag should only lead to visible effects if the two processes are looping at very different rates (e.g., *simControl* is able to do only 5 time steps per second due to massive amount of data saving in the ExS scripts, while *simDisplay* is able to loop 60 times per second).

An alternative is to get *simControl* to compute the LOS value via *GetLOSPosition*(“control”). This will produce a value that is consistent with the physical state of the world, rather than the visual state of the world. It is likely to be “in the near future” of the visual display. The lag ahead is typically on the order of milliseconds, hardly worth worrying about, but can have visible effects (in terms of decisions made by the scripts) if the two processes are looping at very different rates.

Isects are the name give to the result of getting the intersection point between a line and a surface. An isect is said to be “invalid” if there is no such intersection. In the context of AIMSsim, all isects are for a line starting at aircraft location, and having a certain pitch and bearing. Isects typically don’t have a visual component and are useful mostly to the scripts. For this reason, the default *RequestIsect*(*bearing*, *pitch*) get the isect as computed by *simControl*.

Due to the asynchronicity of *simDisplay* and *simControl*, the isection may produce a coordinate that lies on a surface different from what the operator is currently seeing (e.g., the isect result may correspond to the hood of a truck on the ground, but if the truck is moving, the *simDisplay* may not yet have received its new position). In the rare case that this matters, your script can use instead *RequestIsect*(*bearing*, *pitch*, *“display”*) to get it computed by the display. As soon as the display has the result, an `EVENT:ISECT_VALID` is emitted only if a valid isect was found. However, the asynchronicity implies that by the time your script gets run due to the event, the world may already have been evolved by simControl. Typically the change will be small, but there are situations where it could be noticeable.

4.2.7 Timing

There are three concepts of time in AIMSsim:

- **Run time**: time since the beginning of the run, in seconds. Obtained in the experiment scripts by calling *GetRuntime*`()`. This cannot be paused or reset.
- **Simulation time**: time since any motion started, also in seconds. There is motion if **any** of the entities is resumed via an appropriate call to one of the path plan following functions (*PathFollowing**() or *AircraftPathFollowing**() or *TargetPathFollowing**()). This timer pauses when there are no moving entities. Therefore, “pausing the simulation” requires that all entities be paused (e.g., via a call to *PathFollowingPause*()). Therefore if the runtime indicates N seconds since the beginning of the experiment, and the simulation was paused once for P1 seconds and another time for P2 seconds, then the simulation time at exit will be N – P1 – P2. Note that the simulation time is displayed by *simDisplay* in the top right corner of the Operator screen. Note also that the simulation time can be reset to 0 by calling `GenerateEvent(“EVENT:RESET_SIM_TIME”)`.
- **User time**: time since the last *ResetUserTimer*() was called. Obtained in the experiment scripts by calling *GetUserElapsedTime*(). Useful to measure the time differences.

It is important to bear in mind that *AddTimedEvent*() uses the run time, whereas *AddFlightTimedEvent*() uses the simulation time.

4.2.8 Do a task at every time step

If your experiment needs to perform a task at every time step, you can use *AddPeriodicScript*(“scriptName”) to tell *simControl* to execute a script. Periodic scripts can also be removed from the list of periodic scripts. Note that the order of execution of periodic scripts is not guaranteed and should not be relied upon. Note also that adding/removing a periodic script from a periodic script will see the change activated at the *next* time step (so as not to corrupt the list of scripts while it is being traversed).

4.2.9 Events

Events are the main mechanism for indicating to the application that “something has happened”. However, events are useful only if the FSM has transitions defined: the only way to “do something” when an event “occurs”, is to have a transition defined for it in the FSM.

There are several ways of generating events:

- By pressing or releasing a digital control on the input device
- By clicking on a button on a dialog screen on the display

- By calling *GenerateEvent*("EVENT:Name") to generate an event of given name
- By using *AddTimedEvent*() or *AddFlightTimedEvent*(). This will generate the specified event at the specified time: if the event appears in the FSM, a transition could occur. The two timed event queues can be cleared with a call to *ClearEvents*().

Events that are generated during a time step are queued for processing at the next time step, when they will be generated in the same order as they were queued.

Another form of event is specific to *simDisplay*: a call to *SendMessage*("message") sends the given text to *simDisplay*. Note that the order of receipt of messages and events cannot be guaranteed by *simDisplay* because the associated data is communicated to simDisplay over independent channels. A simple example is *UpdateDisplayedTargets*(): this actually uses *SendMessage*(), so the message could be received (in principle) before *simDisplay*'s shared memory has been updated with the new target attributes.

4.2.10 Aircraft motion

The flight plan of the aircraft is set with the *AircraftAddPath*() function, which takes as argument a path plan created with *CreatePathPlan*(). The *AircraftAddPath*() function can be called multiple times to extend the aircraft's flight plan (see the section 4.2.12 on path planning, for a discussion of how path plans are strung together). Wheverever the aircraft reaches the end of the last path given to it, an EVENT:END_OF_PATH:AIRCRAFT is generated.

The motion of the aircraft is enabled and disabled by *AircraftPathFollowing**() functions, where "*" is either *Pause*, *Resume* or *Toggle*.

Note that the motion can also be enabled/disabled by the global *PathFollowing**() functions, which affect all moving entities (aircraft and targets, as appropriate), or a list of specified entities. E.g.

```
PathFollowingToggle()
```

would pause the path following of all entities that were moving, and resume all entities that were paused. Whereas

```
PathFollowingToggle("aircraft", "target one", "target last")
```

would do this only to the specified entities ("aircraft" is the aircraft's "name" as known by the system – it cannot be changed). However, always use one of the *AircraftPathFollowing**() functions instead of calling *PathFollowing**("aircraft").

4.2.11 Targets

Experiments can define targets (*CreateTarget*()) and clear the list of targets (*ClearTargets*()) at any time. It is up to your script to decide if and when to let *simDisplay* know that it should update the display with the new information, via a call to *UpdateDisplayedTargets*() (without arguments).

Each target has attributes that can be modified at any time via the exported *TargetChangeAttrib*() function. As with the creation/deletion functions, your experiment scripts must tell *simDisplay* (when) to update itself. To save time, you can give a sequence of target names as arguments to *UpdateDisplayedTargets*() so *simDisplay* will update only those targets (this cannot be done when creating/deleting targets).

The *TargetChangeAttrib*() function always has as first argument the target's name, the attribute name, and one or more parameters for the new attribute value. Some attributes, such as the target label, require only one such parameter (the label string), whereas others require more: the position, e.g., requires three (time step, new x, and new y). Similarly, *TargetGetAttrib*() returns one, two or three values, based on the attribute type whose value is being queried. E.g. `TargetGetAttrib(“target B”, “pos”)` returns a triplet x,y,z, whereas `TargetGetAttrib(“target B”, “isTarget”)` returns a boolean.

Note that targets get their Z coordinate automatically clamped so the target is on the ground, regardless of its initial Z coordinate value. Currently, flying targets are not possible (unless there is no ground under or above them).

Targets are considered vehicles: they can have motion, just like the aircraft does: *TargetAddPath*(), *TargetPathFollowing**() are available and have the same function as for aircraft, but they apply only to the target specified by name in the first function argument.

When a target reaches the end of the most recently add path plan, the system generates an EVENT:END_OF_PATH:*SUFFIX* event, where SUFFIX is by default the target name (converted to all upper-case letters, and with all spaces converted to underscore). This suffix can be changed via *TargetSetEndOfPathEventSuffix*(). E.g. given a target named “first target”, the default “end of path” event name for that target is EVENT:END_OF_PATH:FIRST_TARGET. After calling *TargetSetEndOfPathEventSuffix*(“first target”, “Group 1”), the next “end of path” event for that target will be named EVENT:END_OF_PATH:GROUP_1.

Note that targets can also be moved manually (e.g., if the ExS scripts define a special path following algorithm) via the *TargetChangeAttrib*() function, i.e. without using any of the path plan following functionality of the system.

4.2.12 Path planning

The concept of path following is available to both types of vehicle entities: aircraft and targets. This allows both the aircraft and targets to follow paths independently. Path following is supported via PathPlan objects created with *CreatePathPlan*(), and the addition of such objects to the vehicles by calling the **AddPath*() function on that vehicle, at any moment during the simulation. Path plans can be sequenced to form more complex plans.

Path plans are considered as objects by the system and are *copied* into the vehicles when the **AddPath*() function is called. Some important consequences of this:

- One path plan can be reused by several different vehicles, including the aircraft
- One path plan can be used several times by the same vehicle
- Modifying a path plan after it has been added to a vehicle does not affect the vehicle's plan

Note however that path plan waypoints contain speed and fillet information. If either of those must change for different vehicles (usually the case if a path plan is going to be used for both the aircraft and some land vehicles (targets), you can copy the points of one plan into another with *PathPlanGetFirstWaypoint*() and *PathPlanGetNextWaypoint*(), applying any transformation desired. E.g.

```
CreatePathPlan("target path")
x,y,z,s,r = PathPlanGetFirstWaypoint("aircraftPath")
while x ~= nil do
    PathPlanAddWaypoint("target 1 path", x,y,z,s/10,r)
    x,y,z,s,r = PathPlanGetNextWaypoint("aircraftPath")
end
```

creates a new path, named "target path", identical to another (already created and populated) path plan named "aircraftPath", but with 1/10th the speed for every waypoint.

Several path plan objects can be added to a vehicle, thus forming a longer, more complex path. Path plans are automatically translated and rotated when they are added to a vehicle that is already on a path, in such a way that the first edge of the added path is collinear with the last edge of the previous plan. In Figure 11 is shown a simple three-waypoint path plan (left), i.e. consisting of two edges. The right side of the figure shows the resulting path plan for a vehicle that has been added the path object twice in a row:

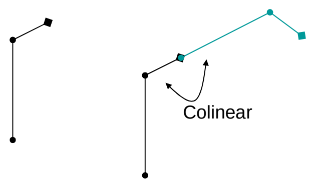


Figure 11. Adding a path a second time to a vehicle

This affects the design of path plans. E.g., in Figure 12, adding a straight path to a target, then adding a loop, then adding the straight path again, leads to radically different overall path for the vehicle, depending on how the loop was designed. The effect in A is typically not what was expected. Notice also how the path objects are rotated to accommodate the colinearity requirement.

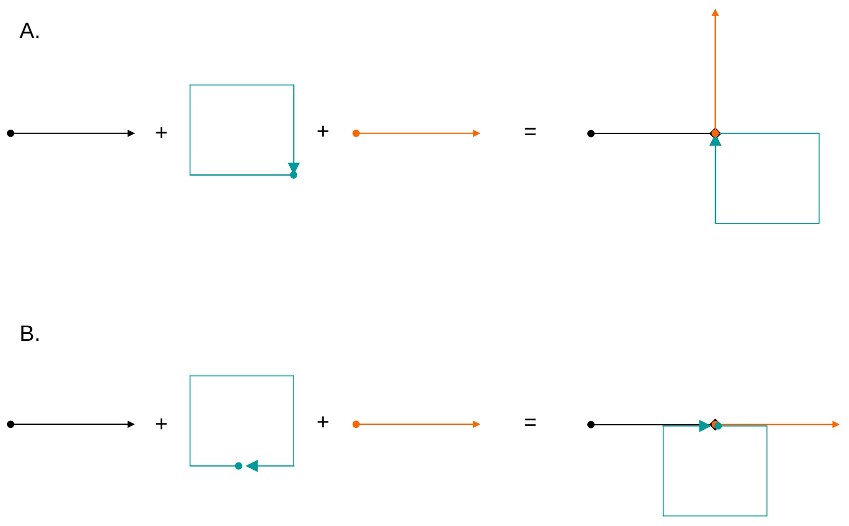


Figure 12. Adding a loop between two copies of a straight path

Path following typically consists in the following steps:

1. Determine which paths are going to be necessary for your experiment;
2. Divide your paths into smaller, reusable "sections", if this helps reduce the number or complexity of paths to create;
3. Determine if and how to react to EVENT:END_OF_PATH:* events;
4. Add the necessary EVENT:END_OF_PATH:* transitions to the FSM, and the associated transition scripts;
5. Create each path section using *CreatePathPlan*("planName"), then calling *PathPlanAddWaypoint*("planName", *x*, *y*, *z*, *speed*, *fillet*) for each point to add to the plan. This can be combined with *PathPlanReadWaypointsFile*("planName", "filename") to add the points from a file. Either way, the points are added as-is (no transformations are applied).
6. Add any combination of the created path plan objects to the aircraft with `AircraftAddPath("planName")`. The first plan added (and only the very first one) sets the initial position of the aircraft to the first point of that plan. The

remaining plans added get translated and rotated so that their first segment is collinear with the previously added plan's last segment (see below).

7. Position targets on the terrain;
8. Add any combination of the created path plan objects to any target with `TargetAddPath("targetName", "planName")`. The first plan to be added gets translated to the target's current location, the remaining plans gets translated and rotated so that their first segment is collinear with the previously added plan's last segment (see below).
9. Resume the appropriate vehicles (aircraft and targets), as desired.

The different default behavior for the first call to *TargetAddPath*() (for a given target) compared to *AircraftAddPath()* shouldn't cause you any surprise, but the following may help clarify further:

> There is only one aircraft, and it will almost always have a flight plan; so the first plan added determines where the aircraft starts and its subsequent motion. However, there are typically many targets, each one created with a specified location, and typically they will not have path plans, and those that do will often use common path object. Therefore it makes sense for path objects to get repositioned onto targets when first added.

The system generates EVENT:END_OF_PATH:* events only when the end of the last path plan added is reached, since the last resume. E.g.

```
AircraftAddPath("aircraftPath")
AircraftAddPath("aircraftPath")
AircraftAddPath("aircraftPath")
AircraftPathFollowingResume()
```

will cause an EVENT:END_OF_PATH:AIRCRAFT event to be generated only after the end of the third path section has been reached.

4.2.13 Save data to a file

The AIMSsim does not create any output files. Rather, output files are created by the experimenter from the Lua scripts using the native Lua output capabilities. However, if having each line of output timestamped is acceptable, then the exported logging facilities are by far the easiest and most robust data saving mechanism. This described in the next section.

Whichever technique you use, AIMSsim does not control where the data goes, how it is formatted in the output file, or any file naming conventions. Some file naming conventions that have worked well, for various types of files:

- Summary file: could contain summary information about the outcome of the experiment. It should end with `_summary.txt` and not involve any streaming data. It could for instance show the duration of the experiment, the tracking score, etc.

- Stream file: contains information that is output at high frequency throughout the experiment, i.e via periodic scripts. E.g. the position and orientation of the simulated search aircraft. It should end with `_stream.txt` extension.
- Log file: contains a history of the important steps or computations during an experiment it shows what happened, in what order. It should end with `_log.txt.`
- Each experiment initialization script creates a "results_P_T" folder in the experiment folder, where P and T are the participant and trial number, respectively. This folder name can be stored in a variable, to be used by all ExS scripts.

4.2.14 Logging

As mentioned in the system manual and in section 3.1, all three of the AIMSsim processes (*simInput*, *simControl* and *simDisplay*) generate their own log file, which gets stored in the folder from which *simControl* is launched i.e. typically from AIMS_HOME. The log files have names of the form "*process*_log.txt" where *process* is one of the three process names e.g. *simControl_log.txt* for *simControl.exe*'s log file. The big advantage of loggers is uniformity of output, time stamping of all messages, and the ability to filter the output based on logger.

All processes use the same logging functionality, which supports the concept of "loggers", identified by a unique character string. Each logger corresponds to a certain "category" of messages, such as *info*, *warn*, *err*, etc (see section 3.1). Each logger can have "sub-categories" of messages. For instance,

- the "err" logger takes care of all error messages from simDisplay;
- the "err.lib" logger takes care of all error messages coming out of application libraries;
- The "err.lua" logger takes care of all messages from simDisplay, specifically related to Lua scripting;
- Etc.

The set of loggers available cannot be changed. However, loggers can be enabled and disabled, and output "destinations" can be added. For simControl, enabling and disabling is done using the two functions *LogEnable*() and *LogDisable*() whereas for *simDisplay*, the two functions are named are *Sim.Loggers.Enable()* and *Sim.Loggers.Disable()*. The loggers of *simInputs* cannot be enabled or disabled. Those functions each take one parameter *loggerName* can contain '*' as wildcard. Examples for *simControl*:

`LogDisable("*")`: disable all loggers!

`LogDisable("info*")`: disable all loggers whose name starts with "info"

`LogDisable("info.lua")`: disable the logger for Lua-specific functions

`LogEnable("*.lua")`: enable all Lua-specific loggers

Adding and removing output destinations is similarly easy. For *simControl*, the two functions to use are *LogAddOutput("output","alias")* and *LogDelOutput("alias")* where "*output*" would be "stdout" or a filename, and "alias" is a string of your choice. For *simDisplay* they are `Sim.Loggers.AddOutput()` and `Sim.Loggers.DelOutput()`, with same argument list. Example for *simDisplay*:

```
Sim.Loggers.AddOutput("*.lua", "file.txt", "file")
Sim.Loggers.DelOutput("*.lua", "file")
```

The first line will cause all output for Lua logs to be duplicated to a file called "file.txt", known as "file". The second line cancels this operation. Each program's default log output file (*_log.txt) was given the alias "`APP_LOG_FILE`".

The system also allows the experimenter to output information from the Lua scripts via the *Log()* function. This is the "User Log" and will have the label "*user_log*" after its time-stamp, rather than "err.lua" or "info.lua". By default, this information appears on the screen only, but as with other log types, it can be duplicated (or moved) to a text file of your choice. The user log is far superior to the native Lua print function since it is time stamped: it can be correlated with the contents of other log output (e.g. to console or other log files).

A table of available loggers is given in Table 10 in Annex E.

4.2.15 Capture display screen

Screen shots of the entire display can be captured by sending a message to *simDisplay*, i.e. *SendMessage*`("CAPTURE_SCREEN")`. The pictures are saved in the experiment's folder, in a subfolder called `results/screens` (created for you if it does not already exist), as a JPEG file of the form `P_T_N.jpg`, where P=participant # (specified on command line, 0 if not), T=trial # (also specified on command line, 0 if not), and N=number of screen shot (N is automatically determined and incremented at every screen grab). Note that image files are overwritten if they already exist (e.g., from a previous run). E.g.., if the experiment is `Designation` (so e.g. the main script is `Designation/main.lua`), then the first screen capture will go to `Designation/results/screens/0_0_0.jpg`, the next one to `Designation/results/screens/0_0_1.jpg`, etc. JPEG files can be viewed with a variety of software, such as Internet Explorer.

4.2.16 Changing the sensor display characteristics

The system makes available five types of sensors. Three EO sensors:

- Electro-Optic Spotter: Narrow FOV color camera (**EO_N** for short). It should have 4 discrete FOV = 0.49°, 0.24°, 0.15°, 0.092°.
- EO color camera: Wide FOV color camera (**EO_W** for short). It should have a continuous zoom with a range of FOV from 1.7° – 20°.

- Active Gated Television: Color camera with laser illuminator and active gating (**AGTV** for short). It should have the same FOV as the EO_N sensor.

And two IR sensors:

- FLIR: standard Forward Looking Infra-Red sensor, FLIR for short. Continuous zoom.
- Thermal Imager: refered to as the "Thermal IR" sensor, THIR for short. It should have 4 discrete FOV = 0.24°, 0.73°, 3.7°, 18.2°.

Each sensor has its own attribute values which can be changed via an exported Lua function, `SetChangeAttrib()`.Table 3 shows the sensor attributes that can be changed via this function. E.g., the second line of the table states that all sensors have a "minFOV" attribute that can be changed, but the middle line shows that only the AGTV has an "illState" attribute, and finally the last line shows that the FLIR doesn't have any specific attributes other than those of the more generic IR and all sensors. Examples:

```
SensorChangeAttrib("EO_N", "minFOV", value)
SensorChangeAttrib("AGTV", "illState", "ENABLED")
```

As shown in this example, the sensor and attribute names as appear in the table are strings, so must be quoted. The type of attribute values (float, enabled/disabled, etc) are the same for all sensors, so the table refers to Table 6 for the description of the attributes and their default values.

Table 3. Sensor attributes available with SensorChangeAttrib()

SENSORS		ATTRIBUTE	VAR DESCRIPTION IN TABLE 6
All:	**AGTV, EO_N, EO_W, FLIR, THIR**		
		`minFOV`	See `agtvMinFOV`
		`maxFOV`	See `agtvMaxFOV`
		`currentFOV`	See `agtvCurrentFOV`
		`degredation`	See `agtvDegredation`
		`LODScale`	See `agtvLODScale`
		`zoomMode`	See `agtvZoomMode`
		`maxZoomIndicator`	See `agtvMaxZoomIndicator`
		`visibility`	See `agtvState`
		`effects`	n/a

	EO:		**`AGTV`, `EO_N`, `EO_W`**	
			`colorDisplay`	See `agtvColorDisplay`
		AGTV:	**`AGTV`**	
			`illState`	See `agtvIllState`
			`illFOV`	See `agtvIllFOV`
			`illNarSize`	See `agtvIllNarSize`
			`illWidSize`	See `agtvIllWidSize`
	IR:		**`FLIR`, `THIR`**	
			`polarity`	See `flirPolarity`
		FLIR:	**`FLIR`**	
			`zoomSlaved`	See `flirZoomSlaved`

Note: Experimenters familiar with older versions of the system will remember the `agtv*` and `flir*` attributes that could be set via the `Set()` function. This is still possible for the AGTV and FLIR for backward compatibility, but new scripts should only change the AGTV and FLIR attributes using `SensorChangeAttrib()` rather than `Set()` since future enhancements (such as additional sensors) will be accommodated through `SensorChangeAttrib()`.

Since there are only two levers on the FlyPanel, one lever controls all EO sensors, and one lever controls all IR sensors. A side effect of this is that if the FLIR zoom is changed while it is visible, then selecting the THIR via the selector bar will display the THIR sensor with the new zoom factor. This is inevitable as the FlyPanel has no feedback mechanism that would allow the zoom lever to be adjusted by AIMSsim based on the selected sensor display. Note that this also means that enabling "zoomSlaved" on the "FLIR" in fact enables it on all IR sensors. The FlyPanel hardware limitations also affect the notion of Primary vs Secondary sensors. Setting "primarySensor" to "PRIM_AGTV" via `Set()` is no longer an accurate statement, since it will in fact set the main viewport (that of EO sensors) as primary, while "PRIM_FLIR" will set the auxiliary viewport as primary.

All sensors use shaders to implement visual effects, such as grayscale, noise, fog, etc. Shaders are regular text files that can be edited via Notepad. E.g. the THIR sensor generates its "visual effect" via the *basicIR.frag* and *basicIR.vert* shader files in the *AIMS_DB/shaders* folder. Those files could be edited by a suitably trained experimenter to change how the THIR display appears. The shader files are identified in Table 4.

Table 4. Shaders used by each sensor display (read from AIMS_DB/shaders)

Sensor name	***Shader files***
AGTV	*agtv.frag, agtv.vert*
EO_N, EO_W	*basicEO.frag, basicEO.vert*
FLIR	*flir.frag, flir.vert*
THIR	*basicIR.frag, basicIR.vert*
AGTV, EO_N, EO_W, FLIR, THIR	*noise2DRed.frag*

4.2.17 Asking for user input: Dialog screens

Dialog screens provide a means to get and react to user input. The system provides three dialog screens by default:

1. an INIT screen which just shows “startup info” such as a copyright notice, software version number, etc.
2. a “Press start” screen that has one button which the user is meant to click in order to “start” the experiment
3. a “Press exit” screen that has one button which the user is meant to click to exit the experiment.

Dialog screens are created in the *simDisplay* configuration file *displayConfig.lua* at startup, as described below. The created screens are then available for display upon request by one of your experiment scripts via a `SendMessage(toDlgName)`. Lastly, your experiment’s FSM determines what happens when a user clicks on a button visible on your dialog screen.

The following text shows an exerpt of a displayConfig.lua script where a new dialog is created and configured:

```
-- Create dialog:
dlg = Sim.CreateDialog("Hello")

-- Add some text to it:
msg = Sim.DialogText("something useful")
dlg:AddText(msg, 0.5, -0.5)

-- Add a button to it:
label = Sim.DialogText("button label")
dlg:AddButton("button1",
    "EVENT:BUTTON_1", label, -0.5, 0.5)
```

The above creates a dialog named “Hello”, adds a text message “something useful” to be positioned at x=0.5, y=-0.5, and then adds a button that will show “button label”, to be centered at x=-0.5, y=0.5 and which will generate the event “EVENT:BUTTON_1” when pressed.

It is worthwhile noting that in this example, one of the predefined system events will be generated when the button is clicked (see Table 7). This implies that the user could also press the physical control button #1 on the FlyPanel to obtain the same effect, since it also generates an “EVENT:BUTTON_1” event. The only restriction on event names is that they start with “EVENT:” and that only system events are capitalized, whereas experimenter event names are not.

All dialog screens use a coordinate system (-1,1) along x and y, hence the center of the screen is at (0,0) whereas the upper right corner is at (1,1). Buttons are positioned from the center, whereas text can be positioned in a variety of ways, the default being “center for x, top for y”. See Figure 13. Refer to Annex B for a list of all modules, symbols and functions available in *displayConfig.lua*.

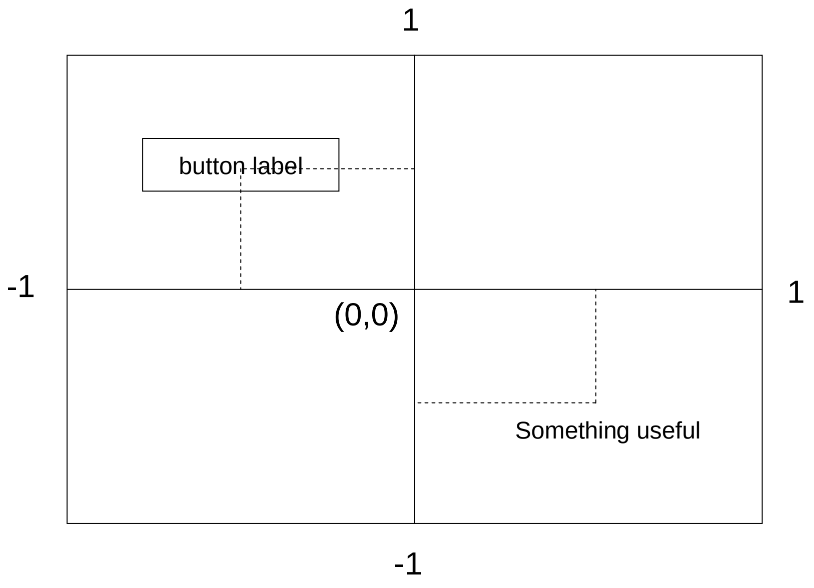


Figure 13. Example dialog screen, also shows coordinate system

Once started, the system can be told to show the created dialog via the command `SendMessage(“toDlgHello”)`. If the experiment’s FSM contains an entry such as

```
AddTransition(“STATE:firstStage”, “STATE:secondStage”,
    “EVENT:BUTTON_1”, “YourExpFolder/configStage2.lua”)
```

then the script *YourExpFolder/configStage2.lua* will be run when the operator clicks on the dialog’s button, assuming the experiment is in the “first stage” state (or transitioning to it) when the `SendMessage(“toDlgHello”)` command is issued. The configStage2.lua script would typically first initiate a SendMessage() to change to a different view (e.g. the Operator View), then take whatever action is appropriate to stage 2 of the experiment (such as saving the operator’s choice, response time, etc).

4.2.18 Using ATD cues

ATD cues can be enabled by creating an “AIMS_ATD_CUES” environment variable before starting the system (this only needs to be done once if defined in the Windows Operating System’s environment). The variable can be set to anything (e.g. 1 or true): the ATD cues will be available if the environment variable exists at start up. Remove it from the environment and restart the application to turn off the ATD cues functionality. Note: setting it to false is not the same as unsetting it, and will not turn the ATD cues functionality off.

Currently there are 3 types of cues: Box, Image, and Color Inversion. During program initialization, the system will go through the list of targets defined and create the specified cues, if any. Cues cannot be created after initialization.

1. ***Box***: The box cue type simply surrounds the target with a red box, as in Figure 14. A target will show this type of cue after an experiment executes the command

```
TargetChangeAttrib(targetName, “atdCueType”, “BOX”).
```

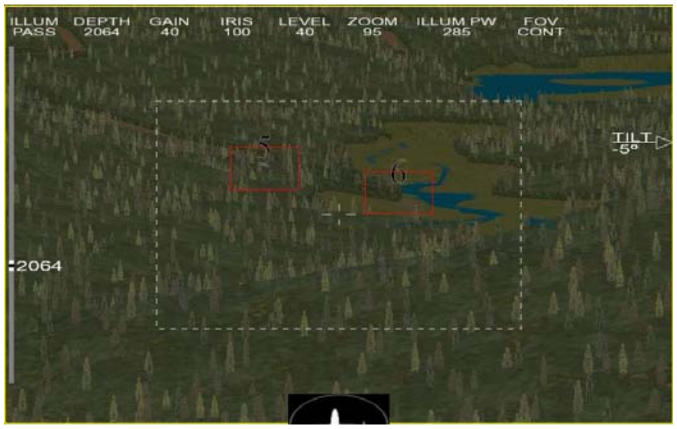


Figure 14: AIMS ATD Cue, *Box* type

2. ***Image***: The image cue type sets a provided image onto the target in a rectangular frameless box, as in Figure 15. The `AIMS_DATA/textures` folder contains an image file conveniently named *default_atd_cue_img.jpg* which is the default image used by Image cues. This file can be replaced with any image of your choice, provided the filename remains the same. Alternately, the environment variable AIMS_ATD_CUE_IMG can be set to the name of a file in `AIMS_DATA/textures`. All targets that use the Image cue type show the same image. A target will show this type of cue after an experiment executes the command

```
TargetChangeAttrib(targetName, “atdCueType”, “IMAGE”).
```

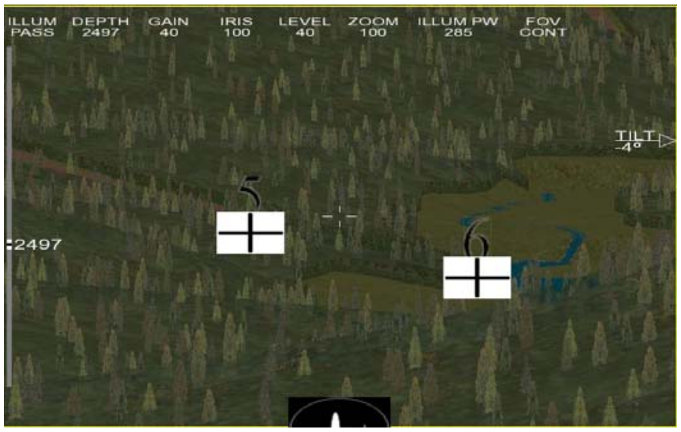


Figure 15: AIMS ATD Cue: *Image* type

3. ***Color Inversion***: This type of cue is represented as an inversion of all colors in the rectangular area centered on the target, as in Figure 16. Only three targets can have this type of cue during one experiment.

```
TargetChangeAttrib(targetName, “atdCueType”, “COLOR_INVERT”).
```

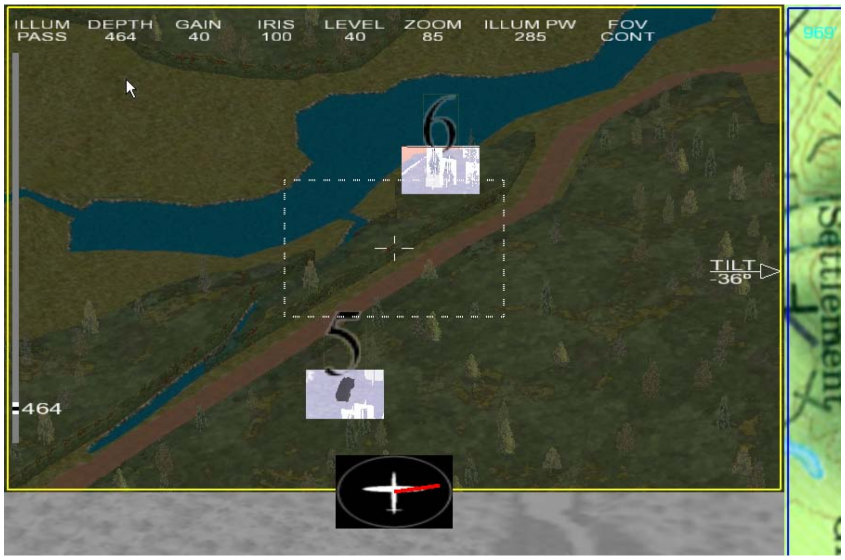


Figure 16: AIMS ATD Cue, *Color Inversion* type

4.3 Example Use of Complete System

In this section we give an example of sequence of steps you might take to create an experiment and run it. The example is based on the Spatial Congruence Experiment.

File definitions and locations:

- `/user/people/elviss/newElviss/bin/NG/SC` – experiment directory
- `scen#.lua` – scenario definition files
- `SC/util` contains:
    - `Base.fp` – flight plan file generated by Scenario Generator
    - `Constants.lua` – includes flying pitch, target pitch, type, and 8 possible bearings; bearings identify speaker positions
    - `Defaults.lua` – AIMSsim settings
- `SC/results` – contains participant log files
- `SC/generic` – contains script files that implement the FSM so no files there should be changed, except `toQuestion.lua` (and only to change wording)

Generating scenario files:

- Copy `scen1.lua` into `scen2-6.lua`
- Open each file and assign it appropriate name (on top) as well as log file name (bottom)

- Set map orientation

Defining target:

- Get appropriate target ID# from `target_map.txt` (in NG) and change type in `constants.lua`
- If new target is to be used, place the file in target directory (to be found out), add it to `target_map.txt`, assign it an ID# and change type in `constants.lua`

Defining target locations:

- 8 target location for each flight direction (E/W)
- 4 legs (2 for each direction) makes 4 targets per leg
- Number of targets has to be a multiple of 16, to have location x flight direction completely crossed
- Number of legs has to be even, to have equal number for each flight direction

Defining flight plan:

- Create flight plan using Scenario Generation Environment (SGE)
- Copy values into `scen#.lua` (might need to set initial aircraft heading or the starting point, not sure about this)
- Leg and turn lengths need to be the same across scenarios, different starting points
- There are only 4 possible starting points: left and right, top and bottom; problem if multiple flight plans used

Generating events:

- Once flight plan is defined, fly over terrain to identify time ranges for each leg during which events can occur
- Leave 5 seconds at beginning/end of each leg to establish RF; also 5 seconds between targets
- Randomly select times for each event within the given ranges

Generating target locations:

- Generate two sets of randomly ordered numbers from 1 to 8, one for each flight direction
- Assign numbers from first set to odd legs, and from second set to even legs (odd: 1-4 and 9-12; even: 5-8 and 13-16)

Defining audio channels:

- Current set-up (`constants.lua`)
    - 1 (target location) = 330 deg
    - 2 = 30
    - 3 = 60
    - 4 = 120
    - 5 = 150
    - 6 = 210
    - 7 = 240

    - 8 = 300
- For N-UP conditions, sound location will be the same as target locations
- For A-UP conditions, sound location will depend on flight direction

Flying E	**Flying W**
1 = 7	1 = 3
2 = 8	2 = 4
3 = 1	3 = 5
4 = 2	4 = 6
5 = 3	5 = 7
6 = 4	6 = 8
7 = 5	7 = 1
8 = 6	8 = 2

This page intentionally left blank.

References

1. Baker, K., & Youngston, G. (2006). *Advanced Integrated Multi-sensor Surveillance (AIMS) Operator-machine interface definition study*. CAE Professional Services. Defence Research and Development Canada - Toronto Contract Report Number: DRDC Toronto CR 2006-242.

2. McFadden, S., Crebolder, J., & Larochelle, V. (2006). *Development of an operator-machine interface for ELVISS Final Report*. Defence Research and Development Canada - Toronto Technical Report Number: DRDC Toronto TR 2006-055.

3. Schoenborn, O. (2007a). *AIMSsim System Manual*. CAE Professional Services, Ottawa, Ontario. Defence Research and Development Canada - Atlantic: DRDC Atlantic CR 2007-301.

4. Schoenborn, O. (2007b). *Human Factors Research Task 2006-111: AIMSsim Feature Development II*. CAE Professional Services, Ottawa, Ontario. Defence Research and Development Canada - Atlantic: DRDC Atlantic CR 2007-302.

Annex A: Exported Scenario Functions and Variables

This section describes all AIMSsim HMI Prototype functions and variables available through the scripting interface, i.e. that have been exported to the Lua interpreter embedded in *simControl*.

A.1 Exported Functions

Table 5. Exported functions and their parameters

Function	Parameters	Description
`Log`	string	Logs string to the currently set log file
`LogEnable/LogDisable`	string `loggerName`	Enable/disable output of log messages for specified logger
`LogAddOutput`	string `loggerName, filename, alias`	Duplicate output going to specified logger into specified file. If file name is “stdout”, duplicates to the console window (from which AIMSsim was started). The alias is optional and defaults to the filename. It can be useful to simplify calls to LogDelOutput(). Note that loggerName can contain wildcards (‘*’).
`LogDelOutput`	string `loggerName, alias`	Stop duplicating specified output (via its aias) for specified logger. Note that loggerName can contain wildcards (‘*’).
`Set`	string `name`, string/number `value`	Sets a system variable to the provided value – see list below
`Say`	string	Deprecated – no TTS App
`PlaySpatialSound`	number `channel`	Deprecated – no sound system
`GetVectorHP`	number `x, y, z`	Returns the heading, pitch for the given vector in world frame. Heading 0 implies North and increases counterclockwise, negative pitch is below horizon..
`GetLOSPosition`	none	Returns the x,y,z position of the terrain in the centre of the sensor image (Line Of Sight) as computed by simDisplay.
`GetLOSPosition`	string `“control”`	Returns the x,y,z position of the terrain in the centre of the sensor image (Line Of Sight) as computed by simControl.
`GetSensorOrientation`	none	Returns the h,p,r orientation of the sensor
`SetSensorOrientation`	number `h,p`	Set new heading (and pitch, if given) of sensor pod. This can only be called if `orientationControl` has been set to `ORIENT_CTL_OPERATOR`
`GetSensorFOV`	none	Returns the Field of View of the sensors (AGTV,FLIR)
`GetSensorZoom`	none	Returns the zoom factor of each sensor (AGTV,FLIR), a value between 0 (full zoom out) and 1 (full zoom in)
`GetViewportInfo`	string `“AGTV”` or `“FLIR”`	Returns the width, height, and dots-per-inch along width and height, for the specified sensor display

`SetDisplayLookAtPoint`	number `x, y, z`	Set the world coordinate that sensor must lock onto if the `orientationControl` is set to `ORIENT_CTL_LOOKAT`
`SetAutoScanInfo`	string `state`, number `center`, number `range`	If first arg is DISABLED, disable any auto-scanning, no more args used; if ENABLED, two more args used: center of scan angle, relative to aircraft, and the scan range, both in degrees
`AddPeriodicScript`	string `scriptName`	Adds a script that is called periodically at high frequency – useful for data collection
`ClearPeriodicScripts`	none	Clears all of the current periodic scripts
`RemovePeriodicScript`	string `scriptName`	Remove a script from list of periodic scripts; does nothing if script not in list
`AddTransition`	string `fromState,` string `toState,` string `triggerEvent,` string `scriptName`	Adds a state transition to help define the experiment finite state machine
`AddTimedEvent`	string `eventName,` number `eventTime`	Adds an event to occur at the simulation time provided; more than one time can be given
`AddFlightTimedEvent`	string `eventName,` number `eventTime`	Adds an event to occur at the flight time provided; more than one time can be given
`GenerateEvent`	string `eventName`	Generates an event
`ClearEvents`	none	Clears all pending timed events (timed, flight timed)
`SendMessage`	string `message`	Sends a control message to the UI – see Table 8
`CreatePathPlan`	string `planName`	Create a new path plan
`PathPlanAddWaypoint`	string `planName`, number `x,y,z,speed,radius, maxAccel`	Adds a waypoint x,y,z to the specified path plan, with desired speed, fillet radius, and maximum acceleration
`PathPlanReadWaypointsFile`	string `planName, fileName`	Read the specified waypoints file (XML file) into the specified path plan, appending the points
`PathPlanGetFirstWaypoint`	string `planName`	Get first waypoint of a plan
`PathPlanGetNextWaypoint`	string `planName`	Get next waypoint of a plan, nil if no more points
`PathPlanClear`	string `planName`	Clears the points of a plan (but does not destroy plan)
`PathFollowingPause`	none, or sequence of entity names	Pauses the path following behaviour of all entities (including aircraft) if no arguments, or of the specified entities if given. Use “aircraft” for the aircraft entity.
`PathFollowingResume`	string `planName`	Same as `PathFollowingPause`, but to resume
`PathFollowingToggle`	string `planName`	Same as `PathFollowingPause` but toggles: resume vehicle motion for every vehicle if paused, and pause if resumed.
`AircraftGetPosition`	none	Returns the x,y,z position of the aircraft
`AircraftGetOrientation`	none	Returns the h,p,r orientation of the aircraft
`AircraftGetVelocity`	none	Returns the vx,vy,vz velocity of the aircraft
`AircraftAddPath`	string `planName`	Copy the specified path object and append to flight plan. If first call, also positions aircraft onto first point of path. For subsequent calls, the path plan copy is translated and rotated so the common point has collinear vertices.
`AircraftPathFollowingResume`	none	Resume motion on path; does nothing if end of

		path reached
`AircraftPathFollowingPause`	none	Pause motion on path
`AircraftPathFollowingToggle`	none	Toggle motion: pause if resumed, resume if paused
`AircraftPrintPath`	none	Dump information about the complete path so far; mostly for debugging the path planning
`ReadTargetsFile`	string `fileName`	Read the specified targets XML file and create targets; this appends targets to current set of targets
`ClearTargets`	none	Removes all targets from the scene
`UpdateDisplayedTargets`	none	Causes simDisplay to update all visual aspects of all targets, including added/removed targets
`UpdateDisplayedTargets`	string `target1Name, target2Name, …`	Causes simDisplay to update all visual aspects of specified targets; only valid for pre-existing targets
`CreateTarget`	string `name,` number `type, x,y,z,h,p,r`	Adds a target to the outside scene with ID name, and type as an index in the `target_map.txt` file. Places target at x,y,z,h,p,r
`TargetChangeAttrib`	string `targetName`, `attribName,` any `newNalue`	Change the specified target's specified attribute to new_value. The type of new_value depends on the type of the attribute. Valid attribute names and associated new value type: `label` string `isTarget` boolean `retroReflective` boolean `colorOverrideAGTV` boolean `colorOverrideFLIR` boolean `colorInAGTV` number, 0..1 `colorInFLIR` number, 0..1 `atdCueType` string
`TargetGetAttrib`	string `targetName`, `attribName`	Get specified attribute from specified target. Valid attribute names and associated value type: `label` string `isTarget` boolean `designationZoneRadius` boolean `pos` x, y, z `vel` vx, vy, vz
`TargetClampGround`	string `targetName`	Clamp given target to ground
`TargetAddPath`	string `targetName`, `planName`	Copy the specified path object and append to vehicle path plan. If first call, translates path onto current target position. For subsequent calls, the path plan copy is translated and rotated so the common point has collinear vertices.
`TargetSetEndOfPathEventSuffix`	string `targetName`, `suffix`	Changes the last part of "end of path" event name for specified target. Default suffix is target's name. Note that suffix is always converted to all-uppercase, and spaces are converted to underscore.
`TargetPathFollowingResume`	none	Resume motion on path; does nothing if end of path reached
`TargetPathFollowingPause`	none	Pause motion on path
`TargetPathFollowingToggle`	none	Toggle motion: pause if resumed, resume if paused

GetRuntime	none	Returns the seconds count since the beginning of the simulation (which is ***not*** the same as the simulation time, the time reported in the top right of the map)
ResetUserTimer	none	Resets the user timer – use to start timing a response
GetUserElapsedTime	none	Returns the elapsed time in seconds since the user timer was last reset
SetInt	string name, number value	Sets the named user variable to the value provided
SetIntArrayElement	string name, number index, number value	Sets the value of the named array at the index provided to the value provided
SetFloatArrayElement	string name, number index, number value	As above, but for floating point numbers
GetInt	string name	Returns the value of the named user variable
GetIntArrayElement	string name, number index	Returns the named and indexed value
GetFloatArrayElement	string name, number index	As above, but for floating point numbers
RequestIsect	number bearing, pitch	Returns validity, x, y, z for a line leaving aircraft with bearing, pitch (given in world frame). Validity is true only if line hits terrain or a target. This is computed by simControl.
RequestIsect	number bearing, pitch, string "display"	Request an intersection computation from aircraft to terrain, at bearing and pitch (given in world frame), but the computation will be done by simDisplay and made available via *GetIsectPos*() after an EVENT:ISECT_VALID.
GetIsectPos	none	Returns the x,y,z location of the last intersection request. Only read in response to an EVENT:ISECT_VALID, or anytime after such an event.
GetParticipant	none	Returns the current participant number
GetTrial	none	Returns the current trial number

A.2 Exported Variables

The exported variables can be changed through the exported *Set*() function, but their state can not be queried.

Table 6. Exported variables, and their type or possible values

Variable Name	Value	Description
baseTerrain	string	The path name of the base terrain database, relative to (and assumed to be present in) AIMS_DATA/terrains
displayConfig	string	The Lua script name for display configuration (defaults to

		displayConfig.lua)
`polarPlotState`	`ENABLED | DISABLED`	Sets visibility of the polar plot, indicating heading of sensor relative to aircraft
`agtvState`	`ENABLED | DISABLED`	Sets the visibility of the AGTV sensor image
`flirState`	`ENABLED | DISABLED`	Sets the visibility of the FLIR sensor image
`mapState`	`ENABLED | DISABLED`	Sets the visibility of the Moving Map Display
`autoTrackState`	`ENABLED | DISABLED`	Tell HMI the state of geo-stabilized tracking mode
`autoAlignState`	`ENABLED | DISABLED`	Tell HMI the state of auto-alignment-to-aircraft-heading
`degredationState`	`ENABLED | DISABLED`	Sets image degradation for the sensors
`timeOfDay`	Float, 0..1	Sets the time of day for the darkness of the AGTV image; 0 = midnight, 1 = noon
`primarySensor`	`PRIM_AGTV | PRIM_FLIR`	Sets the primary sensor to AGTV or FLIR, this affect which sensor footprint displayed in MMD
`fogDistance`	Float >= 0	Sets the fog’s zero visibility distance
`fogOnset`	Float >= 0	Sets the onset distance for the linear fog
`fogColor`	Float, 0..1	Color of fog
`orientationControl`	`ORIENT_CTL_CPU,` `ORIENT_CTL_OPERATOR,` `ORIENT_CTL_LOOKAT`	Sets whether the joystick movements are used to pan camera (sensor displays)
`joystickMode`	`JOY_MODE_AIRCRAFT` `JOY_MODE_CURSOR`	Sets the joystick vertical axes mode
`zoomDirMode`	`ZOOM_MODE_FWD_ZOOM_IN` `ZOOM_MODE_FWD_ZOOM_OUT`	Sets which direction of zoom levers causes zoom in/out
`maxPitch`	Float, -90..90	Sets the maximum pitch constraint on camera
`minPitch`	Float, -90..90	Sets minimum pitch constraint on camera
`mapFile`	string	Sets path for geometry file used if mapMode is MAP_MODE_2D_PAPER, overriding default
`mapAcSymbolState`	`ENABLED | DISABLED`	Sets the visibility of the map aircraft symbol
`mapSensorFPState`	`ENABLED | DISABLED`	Sets the visibility of the map sensor footprint
`mapSensorHistoryState`	`ENABLED | DISABLED`	Sets the visibility of the map sensor history
`mapSensorHistoryRateSquareSize`	Float, >= 0	Sets size of square when adding a point to history; typical values 25m or more
`mapSensorHistoryMaxAdd`	Int, >= 0	Maximum number of history points to add during one frame
`mapMarkingState`	`ENABLED | DISABLED`	Sets the visibility of the map designate markers
`mapFlightPathState`	`ENABLED | DISABLED`	Sets the visibility of the map flight path
`mapNorthIndicatorState`	`ENABLED | DISABLED`	Sets the visibility of the map North indicator
`mapScaleDisplayState`	`ENABLED | DISABLED`	Sets the visibility of the map scale

`mapColour`	`ENABLED | DISABLED`	Sets the map colour mode
`mapOrientation`	`MAP_ORIENT_NORTH_UP | MAP_ORIENT_AIRCRAFT_UP | MAP_ORIENT_SENSOR_UP`	Sets the map orientation behaviour
`mapMode`	`MAP_MODE_2D_PAPER | MAP_MODE_2D_TERRAIN`	Sets the map mode
`mapScale`	float	Sets the map scale desired; assumes map geometry is full-scale
`mapAcSymbolType`	`SYMB_HELO | SYMB_FIXED | SYMB_POINT | SYMB_NONE`	Sets map symbol type
`agtvColorDisplay`	`ENABLED | DISABLED`	Enables/disables the color mode of AGTV (default disabled, i.e. gray)
`agtvIllState`	`ENABLED | DISABLED`	Sets the state of the AGTV Illuminator
`agtvIllFOV`	`ILL_WIDE | ILL_NARROW`	Sets the width of the AGTV illuminator
`agtvIllNarSize`	`SENS_BEAM_NARROW_2 | SENS_BEAM_NARROW_5`	Sets the width of the narrow illuminator
`agtvIllWidSize`	`SENS_BEAM_WIDE_10 | SENS_BEAM_WIDE_15 | SENS_BEAM_WIDE_20 | SENS_BEAM_WIDE_25 | SENS_BEAM_WIDE_35`	Sets the width of the wide illuminator
`agtvZoomMode`	`ZOOM_WIDE | ZOOM_NARROW | ZOOM_CONTINUOUS`	Sets the type of the AGTV zoom to use
`agtvMaxZoomIndicator`	`ENABLED | DISABLED`	Sets the state of the maximum zoom indicator in the AGTV
`agtvMinFOV`	float	Sets the minimum FOV of the AGTV
`agtvMaxFOV`	float	Sets the maximum FOV of the AGTV
`agtvCurrentFOV`	float	Sets the current FOV of the AGTV
`agtvDegradation`	float	Sets the AGTV degradation factor
`agtvLODScale`	float	Sets the AGTV LOD Scale
`flirPolarity`	`POL_BLACK_HOT | POL_WHITE_HOT`	Sets the FLIR polarity
`flirZoomMode`	`ZOOM_WIDE | ZOOM_NARROW | ZOOM_CONTINUOUS`	Sets the state of the FLIR zoom
`flirMaxZoomIndicator`	`ENABLED | DISABLED`	Sets the state of the maximum zoom indicator in the FLIR
`flirZoomSlaved`	`ENABLED | DISABLED`	Enables/disables slaving of FLIR zoom to AGTV zoom
`flirMinFOV`	float	Sets the minimum FLIR FOV
`flirMaxFOV`	float	Sets the maximum FLIR FOV
`flirCurrentFOV`	float	Sets the current FLIR FOV
`flirDegredation`	float	Sets the FLIR degradation factor
`flirLODScale`	float	Sets the FLIR LOD SCALE
`aircraftX (YZHPR)`	float	Sets the aircraft location and orientation
`reqBearing`	float	Sets the requested bearing for the LOS intersection
`reqPitch`	float	Sets the requested pitch for the LOS intersection test
`logFile`	string	Sets the name and path for the log file

Annex B: Display Configuration Commands

This section describes all AIMSsim HMI Prototype modules, functions, classes and variables available for use in the *displayConfig.lua* configuration file. This file is used by the *simDisplay* process at startup, and is read at the same time as *simControl* reads its initialization script.

SimDisplay makes use of Lua in a much more object oriented way than simControl does, thanks to a Lua tool called tolua++. This tool facilitates the export of attributes, functions, methods, and classes and the creation of modules. Lua is based on the concept of tables, hence the following definitions of the above terms:

Function :	Something that can be called, with a *parameter* list. Some functions have optional parameters, and some may return one or more values. Examples: `f(a)`, or `z = f(x,y)`. All *simDisplay* exported symbols are in the `Sim` module.
Module :	A table that can contain *attributes*, *functions*, *classes* (and class instances) and other modules. Some modules are predefined, such as `Sim`, `Sim.OpsDisplay`, and `Sim.Loggers`.
Attribute :	A value, such as a string, integer, instance of a class, etc. An attribute which is not in a table is said to be in the "global" table. Example: `a=1` creates a global attribute 'a' whose value is 1; `Sim.a = 2` creates an attribute 'a' in the `Sim` module.
Object:	Instance of a class, containing *attributes* and *methods*. The instance is typically obtained by calling the class like a *function* with parameters, but can also be created by functions that return instances of the class. E.g. `dialog = Sim.DialogText("some text")` creates a global object named 'dialog' as an instance of the `Sim.DialogText` class, using "some text" as first (and only) parameter to the class's constructor.
Class :	A table which, when "instantiated", represents an *object*. *Instantiation* is done by calling the class like a *function*, with parameters.
Method :	A function specific to an *object*, i.e. to an instance of a *class*. The "function" is called on the object via the following notation: `object:method(params)`.

Sim			
TYPE	**SYMBOL**	**CALL PARAMERS**	**DESCRIPTION**
function	**`CreateDialog`**		**Creates an instance of a `DialogPanel`.**
parameter		*string : name*	Name of dialog
return value		n/a	Instance of `DialogPanel`.
class	`DialogPanel`	n/a	Represent a dialog panel object. It can only be instantiated via `CreateDialog`.
class	**`DialogText`**		**Represent some text to appear on a dialog panel.**
parameter		*string : text*	The text to display
parameter		*float : fontSize*	Size of font to use, as a fraction of dialog screen size; defaults to 0.08.
module	`OpsDisplay`	n/a	Module for configuration specific to the *Operator View*
module	`Loggers`	n/a	Module for configuration specific to the simDisplay's logging functionality
attribute	`TA_`*`LCR_TCBL`*	n/a	A "**T**ext **A**lignment", can be given to DialogText.SetAlignment(). LCRB can be any of LEFT, CENTER, or RIGHT, while TCBL can by any of TOP, CENTER, BOTTOM and BASE_LINE (12 possibilities altogether).
attribute	`TA_BASE_LINE`	n/a	Another "**T**ext **A**lignment".

Sim.DialogPanel			
TYPE	**SYMBOL**	**CALL PARAMERS**	**DESCRIPTION**
method	**`AddText`**		**Add a text item to the dialog panel object**
parameter		*instance : textItem*	An instance of `Sim.DialogText`, containing text to display
parameter		*float : x*	Where to put *textItem*, along *x*; uses *textItem*'s (hidden) alignment attribute to determine how to position it relative to *x*. **Note** that *x* must be in range [-1,1].

parameter		*float : y*	See description for *x*, but along *y*.
method	**`AddButton`**		**Add a button to the dialog panel object**
parameter		*string : buttonID*	Identifier for the button; any unique sequence of characters
parameter		*string : eventName*	The name of the event to be generated when operator clicks button; must start with "EVENT:"
parameter		*instance : label*	An instance of of `Sim.DialogText`, containing text to display inside button
parameter		*float : x0*	Where to position center of button along *x*, on dialog screen. Value must be in range [-1,1].
parameter		*float : y0*	Same as x0, but along *y*

Sim.DialogText

TYPE	SYMBOL	CALL PARAMERS	DESCRIPTION
attribute	string : text		The text string represented
attribute	float : fontSize		The font size to use when displaying this dialog text
attribute	string : font		The font name to use when displaying this dialog text
method	**`SetFont`**		**Change the font used for this dialog text; value will be put in *font* attribute**
parameter		*string : name*	Name of font, preceded by "fonts/". The name can be any font in the OpenSceneGraph *data/fonts* folder. The location of this folder depends on your installation, but is typically *c:\Program Files\OpenSceneGraph\data\fonts*
method	**`SetAlignment`**		
parameter		*int: TA_**	Any of the Sim.TA_* values for text alignment

Sim.OpsDisplay			
TYPE	**SYMBOL**	**CALL PARAMERS**	**DESCRIPTION**
function	**`DivideX`**		**Divide a parent sub-display of Operator View along X axis. Up to five sub-displays can be created with this call. The parameters childName4, parentFraction4, childName5 and parentFraction5 are not shown as they are the same as childName3, parentFraction3. Sum of all fractions must be less than 1.0.**
parameter		*string : parentName*	Name of parent sub-display; can be “ROOT” for root sub-display of Operator View screen
parameter		*string : childName1*	Name of first new sub-display
parameter		*float : parentFraction1*	Fraction of parent’s size, along X, for first child; value in range [0,1]. Value of 0 means the child gets equal share of unused portion of parent area along X.
parameter		*string : childName2*	Name of second new sub-display
parameter		*float : parentFraction2*	Fraction of parent’s size, along X; defaults to 0 (optional) if only two children created
parameter		*string : childName3*	Name of third new sub-display; use only if more than two new sub-displays needed (i.e., optional)
parameter		*float : parentFraction3*	See parentFraction2, but for third child
function	**`DivideY`**		**Same as `DivideX()` but along Y**
parameter		*string : parentName*	See `DivideX()`
parameter		*string : childName1*	See `DivideX()`
parameter		*float : parentFraction1*	See `DivideX()`
function	**`NewChild`**		**Attach a new sub-display of Operator View to a parent sub-display**
parameter		*string : parentName*	See `DivideX()`
parameter		*string :*	Individual name of new sub-display

		childName	
parameter		*float : x0*	Position of center of child along X, on parent; must be in range [-1,1]
parameter		*float : y0*	Same as x0, but along Y
parameter		*float : width*	Fraction of parent width (between 0 and 1)
parameter		*float : height*	Fraction of parent height (between 0 and 1)

`Sim.Loggers`

TYPE	SYMBOL	CALL PARAMERS	DESCRIPTION
function	**`Enable`**		**Enable a logger (or set of loggers)**
parameter		loggerName	Name of logger; may contain wildcard '*' to hit more than one logger, e.g. "err*" will hit all loggers whose name starts with "err".
function	**`Disable`**		**Enable a logger (or set of loggers)**
parameter		loggerName	Same as `Enable()`'s parameter
function	**`AddOutput`**		**Add an output destination to a logger**
parameter		loggerName	Same as `Enable()`'s parameter
parameter		output	Name of output to add. If output="stdout", output is the console window in which *simControl* was started. Otherwise it is assumed to be a valid filename whose contents will be overwritten.
parameter		outputAlias	Alias for the output. Defaults to value of *output*.
function	**`DelOutput`**		**Remove an output destination from a logger**
parameter		loggerName	Same as `Enable()`'s parameter
parameter		outputAlias	Alias for the output, as implied by the call to `AddOutput()`. Does nothing if `AddOutput()` not called first.

Annex C: AIMSsim System Events and Messages

There are several predefined events that are generated by the AIMSsim HMI Prototype under various conditions, and several messages that can be sent to simDisplay to have it perform a task or change what is visible on the display.

Note that “intersection” refers to the intersection between the Line of Sight (LOS) and the ground surface. Such a test yields both a coordinate of the intersection point, and the normal of the polygon containing the intersection point.

Note also that all event names must start with “EVENT:” (not shown in table, for brevity).

C.1 System Events

Table 7. System events and their trigger condition

System Event	**Trigger Condition**
`START_PRESSED`	The start button on the Start Screen is pressed
`EXIT_PRESSED`	The exit button on the Exit Screen is pressed
`ISECT_VALID`	The requested intersection test has been completed
`IMAGE_CAPTURED`	The screen capture request has been completed
`END_OF_PATH:AIRCRAFT`	The aircraft has reached the end of its flight plan
`END_OF_PATH:`*`TARGET_NAME`*	The target entity named TARGET_NAME has reached then end of its path; note that TARGET_NAME is the name of the target, with all letters converted to uppercase, and all spaces converted to underscore
`OPERATIONAL_UPDATED`	The operational view of the display is now fully visible; this gets emitted whenever the display gets a “toOperational” command (via SendMessage())
`BUTTON_TLL`	Top left-most button
`BUTTON_TCL`	Top left-of-center button
`BUTTON_TCR`	Top right-of-center button
`BUTTON_TRR`	Top right-most button
`BUTTON_B??`	Same combinations as for top row, but for Bottom row

`JOYSTICK_TRIG1`	Big trigger on joystick
`JOYSTICK_TRIG2`	Small trigger on joystick
`JOYSTICK_4P_L`	Four-position hat left, on joystick
`JOYSTICK_4P_R`	Four-position hat right, on joystick
`JOYSTICK_4P_D`	Four-position hat down, on joystick
`JOYSTICK_4P_U`	Four-position hat up, on joystick
`JOYSTICK_2P_U`	Mini-hat up, on joystick
`JOYSTICK_2P_D`	Mini-hat down, on joystick
`LEVER_L_TRIG_L`	Left lever's left trigger
`LEVER_L_TRIG_R`	Left lever's right trigger
`LEVER_R_TRIG_L`	Right lever's left trigger
`LEVER_R_TRIG_R`	Right lever's right trigger

C.2 Possible Messages to *SimDisplay* with SendMessage()

The following table outlines the commands that can be sent to the display application using the *SendMessage*() function.

Table 8. Possible messages to simDisplay using SendMessage()

Command	Description
`toStartup`	Displays the Startup Screen
`toOperational`	Enables the "Operator screen" mode of the prototype. This causes event `EVENT:OPERATIONAL_UPDATED` to be generated as soon as the screen becomes visible.
`toDlg`*`Name`*	Display the dialog screen named *Name*. Does nothing if specified dialog doesn't exist. System provides dialogs named "Init", "Startup", and "Exit", other names must refer to dialogs created in the experiment's *displayConfig.lua* script.
`toExit`	Displays the Exit Screen
`MARK_CONTACT`	Marks the Line-of-sight isect as a target contact
`UPDATE_TARGETS`	Updates the displayed targets list

UPDATE_TARGETS_i1_i2_..._iN	Same as UPDATE_TARGETS but update only the targets numbered i1, i2, etc. This message is generated by the system, use *UpdateDisplayedTargets*(t1, t2, … tN) instead, where the arguments are target *names*
REQUEST_ISECT	Requests an LOS intersection test for the bearing and range in the shared memory. It is better to use the *RequestIsect*() function.
CAPTURE_SCREEN	Requests a screen capture. System will generate EVENT:IMAGE_CAPTURED

Annex D: Windows USB Joystick Configuration

Table 9 gives the mapping of the USB joystick's controls to the functionality set by *simControl*. Note that for digital controls, the table also shows the EVENT that can be used in the experiment's Lua scripts (specifically, in the experiment's FSM) to react to the user's input. See Figure 17 for a snapshot of the different controls: A) X and Y axes; B) POV control; C) Button #1 also known as main trigger; D) Button #2 also known as secondary trigger.

Table 9: Windows USB joystick controls mapping

JOYSTICK CONTROL	DESCRIPTION	
X axis	Camera heading	
Y axis	Camera pitch	
Other axes/levers	Unused	
Button 1	Trigger 1 (EVENT:JOYSTICK_TRIG1)	
Button 2	Trigger 2 (EVENT:JOYSTICK_TRIG2)	
Other buttons	Unused	
POV Hat # 1	4-position "hat"; position by angle (degrees):	
	0:	EVENT:JOYSTICK_4P_U
	90:	EVENT:JOYSTICK_4P_R
	180:	EVENT:JOYSTICK_4P_D
	270:	EVENT:JOYSTICK_4P_L
Other POV hats	Unused	

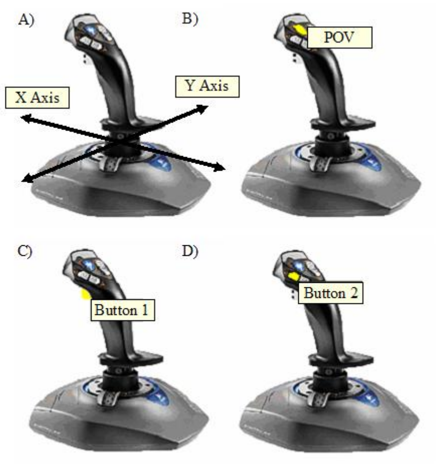


Figure 17. Test Joystick - Logitech Wingman Extreme Digital 3D.

Annex E: Available Loggers

The following table details the loggers used by the AIMSsim processes *simInputs*, *simControl* and *simDisplay*. Note that the default output of all loggers in all three programs has the filename "*prog*_log.txt" where *prog* is the program name. This output has the alias "`APP_LOG_FILE`".

Table 10. Logger names available in AIMSsim

Logger name	**Type of messages**	**AIMSsim processes**
info	Information about "what's happening"; meant to let the experimenter know "what is going on" and what state is the system in.	All three
warn	Warning: "something possibly odd, check this". Output whenever the system thinks something is "odd", but is not otherwise fatal. E.g. if a dialog screen is created and named, a warning is output if the name is already in use: did the experiment intend to replace the existing dialog of same name with the new definition, or did they intend to use a different name which they misspelled? Close attention should be given to warning messages as the system does not halt.	All three
err	Error: something wrong (often – but not always – fatal). An example error might be that a script specified in the initialization script does not exist. Sometimes the system can recover from errors and will continue without stopping, so close attention should be given to err-type messages as well.	All three
info.lua	Like "info", but specific to the interpretation of the experiment scripts, typically to indicate what the system "understands" from the script, thereby allowing the experimenter to compare their intent vs what the system understood.	simControl, simDisplay
err.lua	Like "err", but specific to Lua scripting, e.g. an invalid function name.	simControl, simDisplay
err.lib.*	Same as "err", but generated from one of the libraries used by the program.	All three
user_log	Information printed from script by experimenter, using the `Log()` function (experiment scripts) and `Sim.Log()`	simControl, simDisplay

	function (*displayConfig.lua*)	
dbg.*	Debugging information, which is not output by default. This is useful typically only to the developers. CAE technical support could ask for this to be activated if it cannot reproduce a problem in-house.	All three

Annex F: AIMSsim Target Object Types

Included with the delivery of the AIMSsim Prototype System is a library of target object models. These models may be used as targets/non-targets for the creation and execution of experimental scenarios. The following table provides the names of the target objects, as seen in the old Scenario Generation Environment, and a description of each object model (see Table 12).

Table 11. Target Object Types

NAME	DESCRIPTION
Truck_Desert	Military truck with desert paint scheme
Truck_OD	Military truck with olive drab paint scheme
Hummer_Desert	Military jeep with desert paint scheme
Hummer_OD	Military jeep with olive drab paint scheme
Leclerc_Tank_Desert	LeClerc (French) tank with desert paint scheme
Leclerc_Tank_Camo	LeClerc (French) tank with camouflage paint scheme
Leclerc_Tank_OD	LeClerc (French) tank with olive drab paint scheme
T72_Tank_Desert	T72 (Russian) tank with desert paint scheme
T72_Tank_OD	T72 (Russian) tank with olive drab paint scheme
OH58_Helicopter	OH58 Kiowa reconnaissance helicopter
AH64D_Helicopter	AH64D Longbow apache attack helicopter
A10_Fighter	A10 Warthog Anti-Tank Aircraft
Pyramid_1M	Un-textured 1 meter high pyramid
Pyramid_2M	Un-textured 2 meter high pyramid
Pyramid_3M	Un-textured 3 meter high pyramid
Diamond_1M	Un-textured 1 meter high diamond
Diamond_2M	Un-textured 2 meter high diamond
Diamond_3M	Un-textured 3 meter high diamond
Cube_1M	Un-textured 1 meter high cube
Cube_2M	Un-textured 2 meter high cube
Cube_3M	Un-textured 3 meter high cube

Cylinder_1M	Un-textured 1 meter high cylinder
Cylinder_2M	Un-textured 2 meter high cylinder
Cylinder_3M	Un-textured 3 meter high cylinder
Pyramid_1M_TEX	Textured 1 meter high pyramid
Pyramid_2M_TEX	Textured 2 meter high pyramid
Pyramid_3M_TEX	Textured 2 meter high pyramid
Diamond_1_TEX	Textured 1 meter high diamond
Diamond_2M_TEX	Textured 2 meter high diamond
Diamond_3M_TEX	Textured 3 meter high diamond
Cube_1M_TEX	Textured 1 meter high cube
Cube_2M_TEX	Textured 2 meter high cube
Cube_3M_TEX	Textured 3 meter high cube
Cylinder_1M_TEX	Textured 1 meter high cylinder
Cylinder_2M_TEX	Textured 2 meter high cylinder
Cylinder_3M_TEX	Textured 3 meter high cylinder
Murray	“Murray” person figure
Homer	“Homer Simpson” person figure

Annex G: Converting .flt files to .ive

OpenSceneGraph accepts a variety of input file formats for geometry. The fastest format to load is a binary format that has the “ive” extension. The program osgconv.exe can be used easily to convert any of the file formats understood by OpenSceneGraph to any other format. E.g., from your command shell you could do

```
>> osgconv truck.flt truck.ive
```

to convert the truck geometry stored as an OpenFlight format into OpenSceneGraph’s binary format. The above assumes you have downloaded and installed osgconv from the OpenSceneGraph web site (www.openscenegraph.org), that osgconv.exe is in your PATH environment variable, and that the truck.flt geometry and its textures are in the current folder.

Annex H: AIMSsim Scenario Generation Environment

H.1 Introduction

The AIMSsim Scenario Generation Environment (SGE) allows you to define the elements required to create an experimental scenario. The SGE also allows you to preview some elements of the scenario as it is being created. Specifically, you can:

- Define a scenario terrain database for the simulated environment.
- Place target objects into the simulated environment.
- Define a flight path for a simulated search aircraft.
- Define moving map characteristics.
- Adjust various simulated sensor characteristics.
- Adjust simulated environmental conditions.
- Configure the layout of the Human Machine Interface (HMI).

The SGE uses a project concept. An SGE project is a collection of all the files that together make up an experimental scenario. You begin by planning your project and then create various files as you perform the tasks required to build your experimental scenario. The SGE uses four files to define a scenario: a target file, a flight plan file, a configuration file and a project file.

- The target file (*targets.xml*) contains the information required to represent the placement and characteristics of all targets for a given scenario.
- The flight plan file (*flightplan.xml*) contains the information required to create a flight path for a simulated search aircraft.
- The configuration file (*config.xml*) contains various elements of scenario configuration information including a definition of the Scenario Landscape, the configuration of the simulated sensors and the configuration of the HMI Prototype.
- The project file (*project.xml*) contains a reference to a target file, a flight plan file and a configuration file which, when treated as a group, represent a complete experimental scenario.

Note however that only two of the files (i.e. *targets.xml* and *flightplan.xml*) produced by the SGE are usable by the AIMSsim HMI Prototype, a significant departure from the old AIMSsim system. The scenario development process, if you wish to you use the SGE, is therefore as sketched in Figure 18.

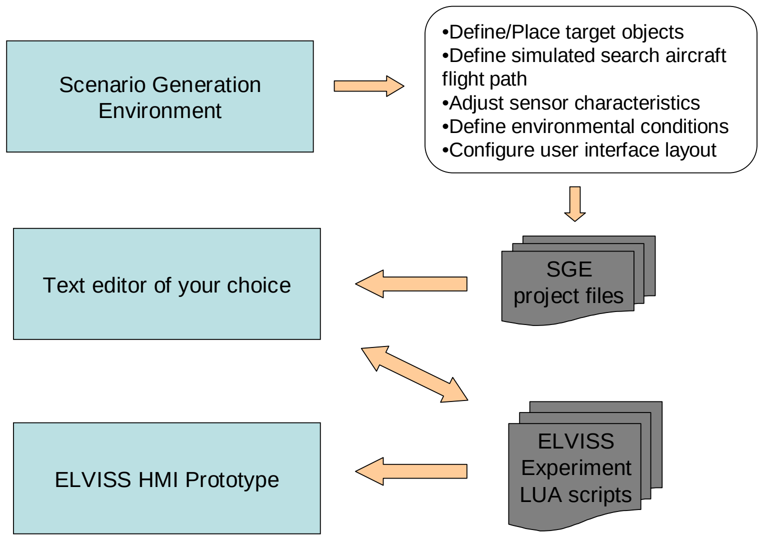

Scenario Generation Environment
•Define/Place target objects
•Define simulated search aircraft flight path
•Adjust sensor characteristics
•Define environmental conditions
•Configure user interface layout
Text editor of your choice
SGE project files
ELVISS HMI Prototype
ELVISS Experiment LUA scripts

Figure 18. AIMSsim experiment development process when it involves the SGE

H.2 Starting and Exiting the SGE

To start the SGE type **scenGen** in a command prompt window open to the location of the SGE executable file. This will load the SGE. You may also double-click on this executable file on your desktop or from the Windows Explorer application wherever it happens to reside on your local/network drive.

To exit the SGE, choose **File | Exit**.

H.3 A quick look at the SGE

Once you start the SGE, you will use various controls to perform your scenario-building tasks:

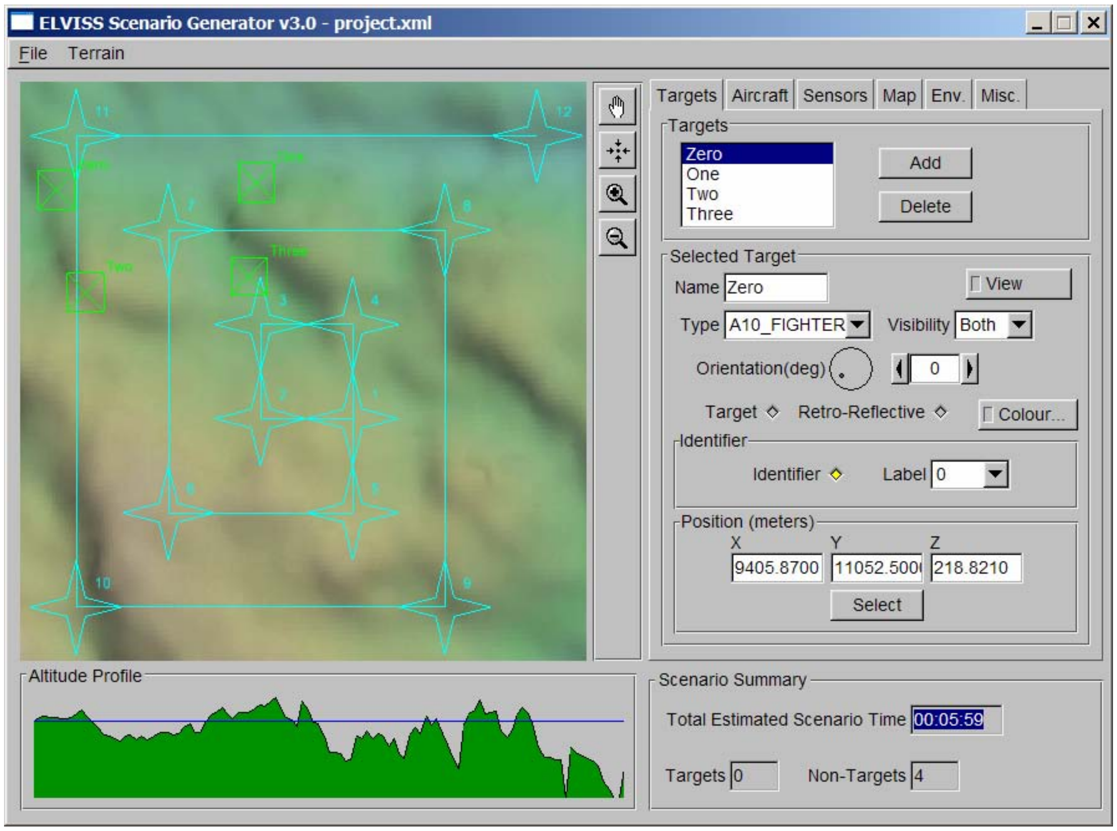
- The **Moving Map Display** (MMD) and **Toolbar** allow you to navigate the terrain and scrutinize the placement of objects. You may also view a 3D display of the object and its placement in the terrain by pressing the 'view' button.
- The **Targets** tab allows you to add and delete targets from the scenario as well as to control the characteristics of the targets.
- The **Aircraft** tab allows you to define a flight path for a simulated search aircraft.
- The **Sensors** tab allows you to control the characteristics of the simulated AGTV and FLIR sensors.
- The **Map** tab allows you to select map characteristics.
- The **Environment** tab allows you adjust environmental settings for the scenario.
- The **Misc** tab allows some additional settings to be selected.

- The **Altitude Profile Display** provides you with information about the altitude of the simulated search aircraft with respect to time.
- The **Scenario Summary** provides you with summary information about the duration of the current scenario and the number of targets/non-targets defined in the current scenario.

The SGE also includes various menus, each with its own set of commands and/or options that you can use to perform functions such as loading/saving scenarios and files and selecting the scenario landscape.

H.4 The SGE Interface

The SGE interface consists of a single window that is divided into six main areas and a menu bar, as illustrated in Figure 19. The options provided in the pull-down menus are described in the following sections.



ELVISS Scenario Generator v3.0 - project.xml

Targets: Zero, One, Two, Three

Selected Target

Name: Zero

Type: A10_FIGHTER

Visibility: Both

Orientation(deg): 0

Target ◇ Retro-Reflective ◇

Identifier

Identifier ◇ Label: 0

Position (meters)

X	Y	Z
9405.8700	11052.500(	218.8210

Total Estimated Scenario Time: 00:05:59

Targets: 0 Non-Targets: 4

Figure 19. SGE Interface

H.5 Using the AIMSsim SGE

The following sections will provide information about using the SGE to perform the standard tasks required to build AIMSsim experimental scenarios.

1. Defining a Scenario Landscape (Terrain)

Each AIMSsim experimental scenario is based on a terrain database. To open a database, use **Terrain | Select…** in the SGE window. Terrain database files may be recognized by the .ive file extension. Navigate to the appropriate directory and select the file that you want to open, and click **OK**. The selected terrain database appears in the Moving Map Display (MMD) area.

The terrain databases that are supplied with AIMSsim include: the Certain Impact database and the Nerepis database. To navigate “certain_impact/models” select “terrain.pfb”. To use the Nerepis database, follow the instructions in the following paragraph.

When selecting a terrain in the Nerepis database select **only:** NE, SW, NW, or SE. When navigating the directories that contain terrain data you will find multiple files: one with “_bw”, “_shaded”, or “shaded_bw” in the name and one without (i.e. terrain.pfb and terrain_bw.pfb). The file name that contains “_bw” is the black and white version of the terrain database – this version **should not** be used in the SGE neither should “_shaded”, nor “shaded_bw”. Instead, use the file without “_bw” in the file name – this is the colour version of the terrain database and **should** be used in the SGE.

2. Using the Moving Map Display

The SGE allows you to navigate the Scenario Landscape via the Moving Map Display (MMD) which is located in the box on the left hand side and associated Toolbar. The main purpose of the MMD is to provide a visual overview of the experimental scenario and to facilitate accurate placement of scenario elements (targets and waypoints) on the Scenario Landscape.

a. The MMD Toolbar

The MMD Toolbar provides additional tools to facilitate the manipulation of the MMD (see Figure 20). The Toolbar provides such functions as *Grab*, *Centre On*, *Step Zoom In* and *Step Zoom Out*.

The Grab tool provides an easy means to navigate the Scenario Landscape by allowing the user to grab and drag the terrain in any direction. To use the Grab tool, simply click on the Grab tool icon. Now use the left mouse button to select a drag point on the terrain and with the left mouse button depressed, move the mouse in any direction to drag the terrain. The Grab tool mode is persistent, so when you have finished repositioning the terrain, de-select the grab tool by clicking on the Grab tool icon.

The Centre-On tool allows you to select a point on the Scenario Landscape that you wish to specify as the new “centre” for the MMD. To use the Centre-On tool, click the Centre-On tool icon. Use the mouse to position the cursor over the position that you would like to specify as the new centre for the MMD and depress the left mouse button to re-centre the map. The Centre-On tool mode is also persistent, so when you have finished specifying a new centre, you will have to de-select the Centre-On tool. The Centre-On function is particularly useful when you wish to quickly zoom in on a specific position on the terrain: rather than having to use a combination of Grab and Zoom actions simply centre on the area of interest and use the Step Zoom functions to achieve the desired zoom level.

The Step Zoom In and Step Zoom Out tools allow you to quickly zoom in and out on the MMD in an incremental fashion. To Step Zoom In: click on the Step Zoom In icon.

To Step Zoom Out: click on the Step Zoom Out icon. The zoom function will halve the current zoom setting or double the current zoom setting, as appropriate.

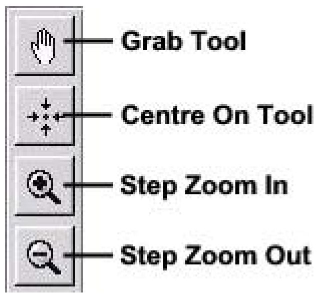
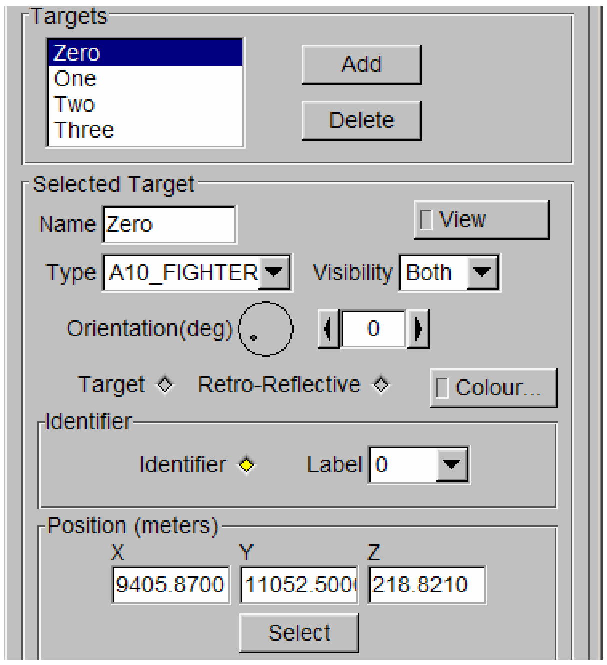


Figure 20. *MMD Toolbar Functions*

3. Manipulating Targets

Target objects form the basis for the search and detection task utilized in the empirical investigation of AIMSsim user interface issues. A target is a three-dimensional object or 3D model with certain scenario related characteristics assigned to it by the user. Targets may be added or deleted from a scenario. The target manipulation interface is presented in Figure 21.



Name: Zero
Type: A10_FIGHTER
Visibility: Both
Orientation(deg): 0
Label: 0
Position (meters) X: 9405.8700 Y: 11052.500(Z: 218.8210

Figure 21. *Target Manipulation Interface*

a. Adding a Target

In the tabbed area select **Targets**. The target manipulation area will be displayed.

In the Targets area, add a target by clicking the **Add** button. A new target will appear in the target browser. It will have been assigned a default name of “Target”.

At this point a target icon will not have appeared on the MMD. This is because no position has been defined for the newly created target. To define a geographic position for the target select a location on the Scenario Landscape by clicking on the MMD in the location where you’d like the target to be placed and then click the **Select** button in the Position area.

A target icon will appear on the MMD over the clicked position to indicate the selected location for the target. Target icons appear as boxes containing an “x” symbol”.

In order to achieve the most accurate placement of target objects, you should zoom in to the desired placement area on the MMD before selecting the target location.

b. Deleting a Target

1. In the tabbed area, select **Targets** if it is not already selected. The target manipulation area will be displayed.
2. Select the target you wish to delete via the target browser. It will become highlighted.
3. Delete the target by clicking the **Delete** button.

c. Modifying Target Parameters

Each target has a number of parameters that will affect the way the target will be utilized by the AIMSsim HMI Prototype. The following table describes the various target parameters and how to manipulate them.

Table 12. *Target Parameters*

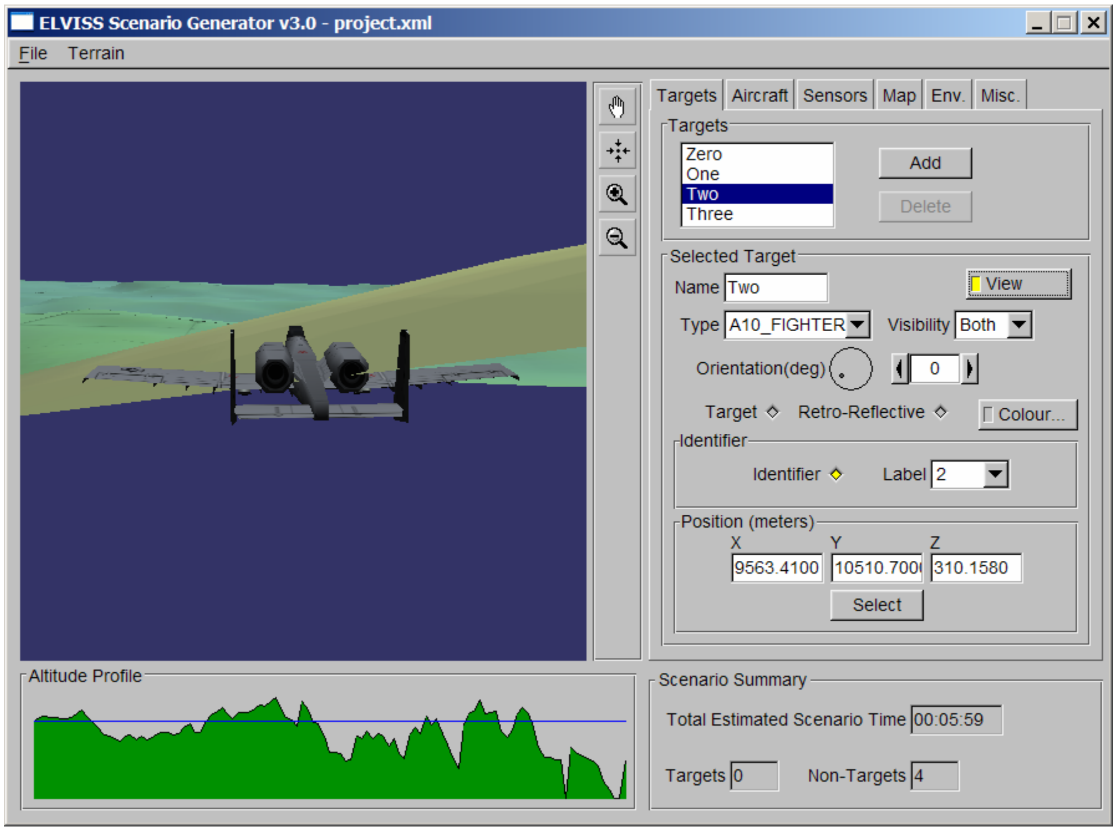
PARAMETER NAME	DESCRIPTION	USE
Name	Assigns an alpha-numeric name to the target. The name is used to identify the target on the MMD.	Select the **Name** field and type a desired name. When finished, press the <Enter> key to apply the new name.
Target	Flags this object as being a “Target” object (as opposed to a non-target). This parameter is used for scoring purposes when an HMI Prototype use has been asked to discriminate between true target objects and false target objects.	Click on the **Target** radio button to toggle the target setting. Notice that the colour of the target icon on the MMD will toggle between red and green. Red indicates that the target has been flagged as a true target. Green signifies a false target.

Retro-Reflective	Flags this object as having “retro-reflective” characteristics. This parameter will affect the presentation of the target when observed in the HMI Prototype: the target will appear to “reflect” the light emitted by the laser illuminator when the beam is positioned on the target.	Click on the **Retro-Reflective** radio button to toggle the retro-reflective setting. Illuminated indicates retro-reflective property enabled.
Type	Defines the visual representation of the target (people, vehicles, aircraft or geometric primitives).	Select a type from the **Type** pick list. See Annex F for descriptions of the targets.
Visibility	Defines the visibility of target with respect to the simulated AGTV and FLIR sensors.	Select a visibility setting from the **Visibility** pick list. **Both** signifies the target will be visible to both the simulated AGTV and FLIR sensors. **AGTV Only** means that the target will be visible to the simulated AGTV only and will not be visible by the FLIR. **FLIR Only** means that the target will only be visible to the simulated FLIR sensor and will not be visible to the AGTV sensor.
Orientation	Assigns an orientation or heading to the target object allowing it to be rotated to face in any direction. A heading of 0 is equivalent to due North.	Adjust the **Orientation** dial to specify the desired orientation of the object. Alternatively, you may increment or decrement the orientation value via the **Orientation** spin box.
Colour	Allows you to modify the runtime colour of the target object as it will appear to the AGTV and FLIR sensors. The light on the **Colour** button determines if the target colour has been modified (lit) or if the default target colour is being used (unlit).	Click the **Colour** button to display the target colour manipulation window. Use the **Enable Colour Adjustment** radio buttons to enable/disable colour adjustment for the current target as it will be viewed by the AGTV and FLIR sensors. When colour adjustment is enabled, use the slider to adjust the target colour between **Dark** and **Light**. When you have finished adjusting target colour settings, click **OK** to keep you changes or **Cancel** to discard any changes you have made.
Identifier	Flags this object as being enhanced by an identifier label. An identifier label takes the form of a billboard “sign” that will be presented above the target model representation when viewed in the HMI Prototype.	Click the **Identifier** radio button to select/deselect the target identifier setting.
Label	When the **Identifier** is enabled, **Label** defines the alpha-numeric character that will be visible on the identifier billboard.	Select an alpha-numeric value from the **Label** pick-list. This parameter will only have an effect if the **Identifier** parameters have been enabled.
Position	Assigns the geographic location of the target object on the scenario landscape. The position is specified by an x, y, and z value. These values are in meters.	The target position may be selected interactively by clicking the **Select** button and then clicking on a location on the MMD. Alternatively, you may select and type discrete values into the **x**, **y** and **z** fields and then hit <Enter> to apply the new value(s).

d. Previewing Target Placement and Appearance

The SGE allows you to view the placement and appearance of the targets that you have created. This feature is particularly useful when accurate placement of objects with respect to the surrounding terrain is desired (i.e. placing an object between two buildings, behind a hill, etc.). The view, however, does not reflect the weather degradations and the colour level of the targets.

To view a target, click on the target name in the target browser and press the ‘view’ button to toggle the target view mode. A 3D view of the target and its surrounding terrain will become visible in the MMD. The target view mode is presented in Figure 22.



Targets: Zero, One, Two, Three

Selected Target

Name: Two

Type: A10_FIGHTER

Visibility: Both

Orientation(deg): 0

Target ◇ Retro-Reflective ◇

Identifier

Identifier ◆ Label: 2

Position (meters)

X: 9563.4100 Y: 10510.700(Z: 310.1580

Scenario Summary

Total Estimated Scenario Time: 00:05:59

Targets: 0 Non-Targets: 4

Figure 22. Target View Mode

In target view mode, the selected target and the surrounding scene can be navigated by using the left and right mouse buttons. Using the left mouse button allows the user to rotate the target and the scene to a number of angles and views. The right mouse button allows the target and scene to be zoomed both in and out. When you are finished viewing the target, press the ‘view’ button again to toggle out of target view mode.

e. Saving the Target List

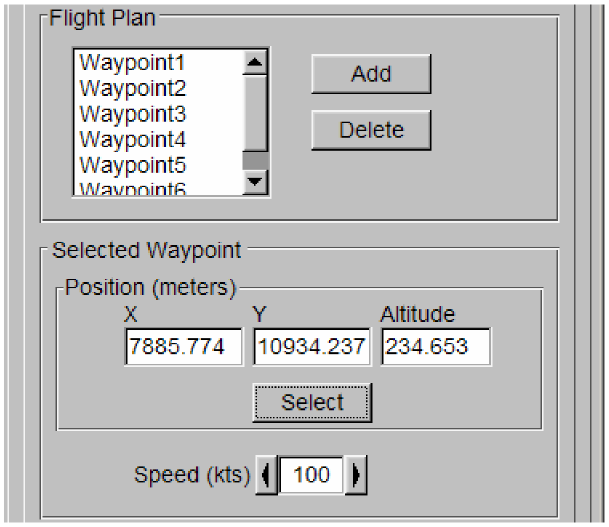
Whenever you make changes to your targets you will want to preserve those changes by saving the target list. To save the target list, select **File | Save Targets** on the SGE window menu bar. Use the file browser to navigate to the desired directory. The target list and related target properties will be saved to the 'targets.xml' file within the chosen directory. Click **OK** to save the target file. Alternatively you may wish to overwrite an existing target file. In this case use the file browser to navigate to the directory that already contains the 'targets.xml' file that you wish to overwrite and click **OK**.

4. Manipulating the Aircraft Flight Path

In order to represent the use of AIMS from an airborne platform, the AIMSsim Prototype System allows you to define a flight path for a simulated search aircraft to which the simulated sensors are affixed. The SGE allows you to select one of three types of flight paths: a User Defined flight path, a Creeping Line Ahead flight pattern and an Expanding Square flight pattern.

a. User Defined Flight Path

The User Defined flight path allows you to create a flight path made up of a sequence of waypoints that are individually placed according to your specifications. You may specify the location, altitude and speed of the simulated search aircraft at each waypoint. The User Defined flight path manipulation interface is presented in Figure 23.



X: 7885.774
Y: 10934.237
Altitude: 234.653
Speed (kts): 100

Figure 23. *User Defined Flight Path Manipulation Interface*

b. Adding a Waypoint

1. In the tabbed area select **Aircraft** then select the **User Defined** tab. The User Defined flight path manipulation area will be displayed.
2. In the Flight Plan area, add a waypoint by clicking the **Add** button. A new waypoint will appear in the Waypoint Browser. The SGE will automatically assign a name to the waypoint (i.e. "Waypoint1").

3. At this point a waypoint icon will not have appeared on the MMD. This is because no position has been defined for the newly created waypoint. To define a geographic position for the waypoint, select a location on the Scenario Landscape by clicking on the MMD in the location where you'd like the waypoint to be placed and then click the **Select** button in the Position area.
4. A waypoint icon will appear on the MMD over the clicked position to indicate the selected location for the waypoint. Waypoint icons appear as cyan coloured stars. Once more than one waypoint has been defined, the flight path will appear on the MMD as a cyan line connecting waypoint to waypoint.
5. The Altitude Profile Display provides you with information about the relative altitude of your waypoint from ground level.

In order to achieve a more accurate placement of waypoints, you may want to zoom into the desired placement area on the MMD before selecting the waypoint location.

c. Deleting a Waypoint

1. In the tabbed are select **Aircraft** if it is not already selected. The flight path manipulation area will be displayed.
2. Select the waypoint you wish to delete via the Waypoint Browser. It will become highlighted.
3. Delete the waypoint by clicking the **Delete** button. Any remaining waypoint will automatically be re-named to ensure a continuous sequence of numbers in the flight plan.

d. Modifying Waypoint Parameters

Each waypoint has a number of parameters that will affect the way the waypoint will be utilized by the AIMSsim HMI Prototype. The following table describes the various waypoint parameters and how to manipulate them.

Table 13. *User Defined Waypoint Parameters*

PARAMETER NAME	DESCRIPTION	USE
Position	Assigns the geographic location of the waypoint object on the scenario landscape. The position is specified by an x, y, and Altitude value. These values are in meters.	The waypoint position may be selected interactively by clicking the **Select** button and then clicking on a location on the MMD. Alternatively, you may select and type discrete values into the **x**, **y** and **Altitude** fields and then hit <Enter> to apply the new value(s).

Speed	Assigns a speed to the simulated search aircraft once it reaches that waypoint. Speed is specified in knots.	Click on the increment or decrement controls on the **Speed** spin box to increase or decrease the speed of the simulated search aircraft.

e. Creeping Line Ahead Flight Pattern

The SGE allows you to generate flight patterns that are commonly employed by the search and rescue community. The “Creeping Line Ahead” is one of these patterns. The “Creeping Line Ahead” comprises a pattern of equally spaced parallel lines. It is a general search pattern that attempts to ensure even search coverage over a designated search area. A “Creeping Line Ahead” pattern is depicted in Figure 24.

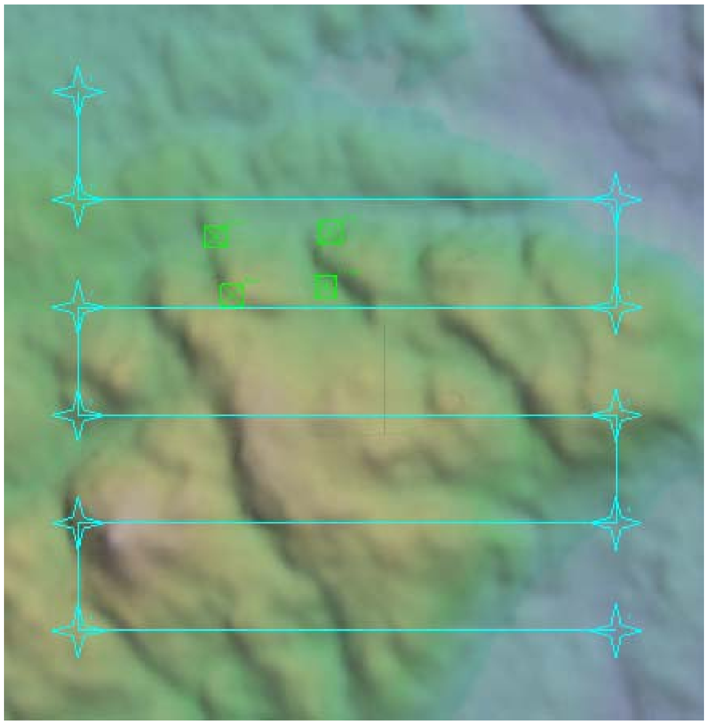


Figure 24. *Creeping Line Ahead Search Pattern*

By providing some elementary parameter values, the SGE will automatically generate a sequence of waypoints for you.

f. Creating a Creeping Line Ahead Flight Pattern

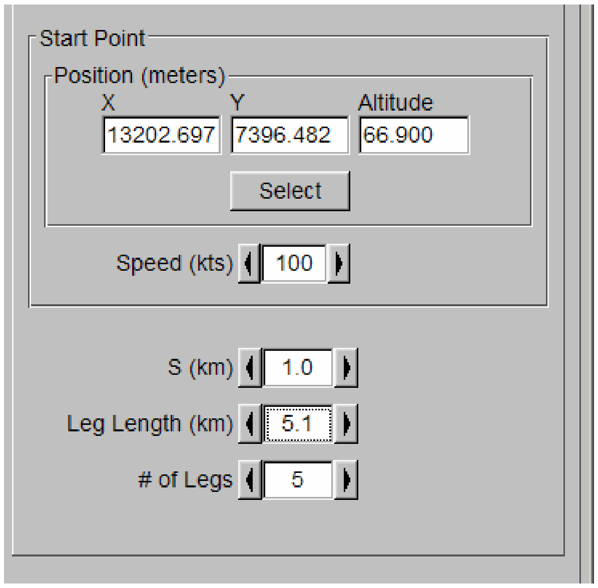
To create a Creeping Line Ahead flight pattern:

1. In the tabbed area select **Aircraft** then select the **Creeping Line** tab. The Creeping Line Ahead flight path manipulation area will be displayed.
2. In the Start Point area, designate the start point for the pattern by clicking a location on the MMD and then pressing the **Select** button. The

Creeping Line Ahead flight pattern will be displayed on the MMD configured with default parameter values.

g. Modifying the Creeping Line Ahead Flight Pattern

The Creeping Line Ahead flight pattern is created based on a number of parameters. Changing these parameters will affect the geographic area that the pattern will cover and the rate at which the area is covered. The following table describes the various parameters that affect the Creeping Line Ahead flight pattern and how to modify them. The Creeping Line Ahead manipulation interface is depicted in Figure 25.



X: 13202.697
Y: 7396.482
Altitude: 66.900
Speed (kts): 100
S (km): 1.0
Leg Length (km): 5.1
# of Legs: 5

Figure 25. *Creeping Line Ahead Flight Pattern Manipulation Interface*

Table 14. *Creeping Line Ahead Parameters*

PARAMETER NAME	DESCRIPTION	USE
Position	Assigns the geographic location of the starting point for the flight pattern. The position is specified by an x, y, and Altitude value. These values are in meters.	The Start Point position may be selected interactively by clicking on a location on the MMD and then pressing the **Select** button. Alternatively, you may select and type discrete values into the **x**, **y** and **Altitude** fields and then hit <Enter> to apply the new value(s).
Speed	Defines the speed at which the simulated search aircraft will fly the flight pattern.	Click on the increment or decrement controls on the **Speed** spin box to increase or decrease the speed of the simulated search aircraft.

S	Represents the separation between the flight legs that make up the pattern. S is measured in kilometres.	Click on the increment or decrement controls on the **S** spin box to increase or decrease the leg separation. This value should be at least 0.1 km.
Leg Length	Represents the length of the flight legs that make up the pattern. Leg Length is measured in kilometres.	Click on the increment or decrement controls on the **Leg Length** spin box to increase or decrease the length of the flight legs. This value should be at least 0.1 km.
# of Legs	Assigns the number of flight legs that will make up the flight pattern.	Click on the increment or decrement controls on the **# of Legs** spin box to increase or decrease the number of legs that make up the pattern.

Figure 26 describes the Creeping Line Ahead flight pattern and the parameters used to generate it.

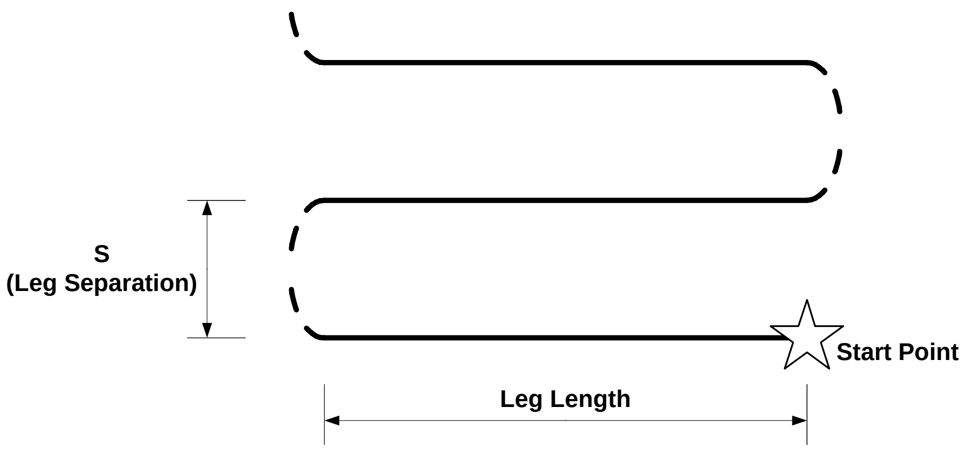


Figure 26. *Creeping Line Ahead Parameters*

h. Expanding Square Flight Pattern

The “Expanding Square” is another commonly used search pattern. It is made up of a pattern of progressively larger squares (a ``square spiral''). The “Expanding Square” flight pattern is a more specialized search pattern employed when the general location of the object or person being searched for is known and the search crew wishes to concentrate around that area. An “Expanding Square” flight pattern is depicted in Figure 27. By providing some elementary parameter values, the SGE will automatically generate a sequence of waypoints for you.

i. Creating an Expanding Square Flight Pattern

To create an Expanding Square flight pattern:

1. In the tabbed area select **Aircraft** then select the **Expanding Square** tab. The Expanding Square flight path manipulation area will be displayed.
2. In the Start Point area, designate the start point for the pattern by clicking a location on the MMD and then pressing the **Select** button. The Expanding Square flight pattern will be displayed on the MMD configured with default parameter values.

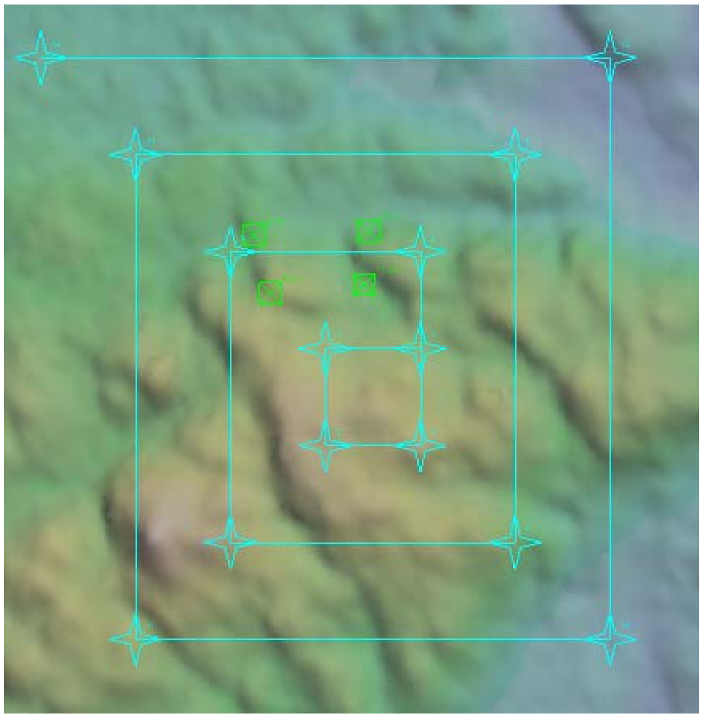
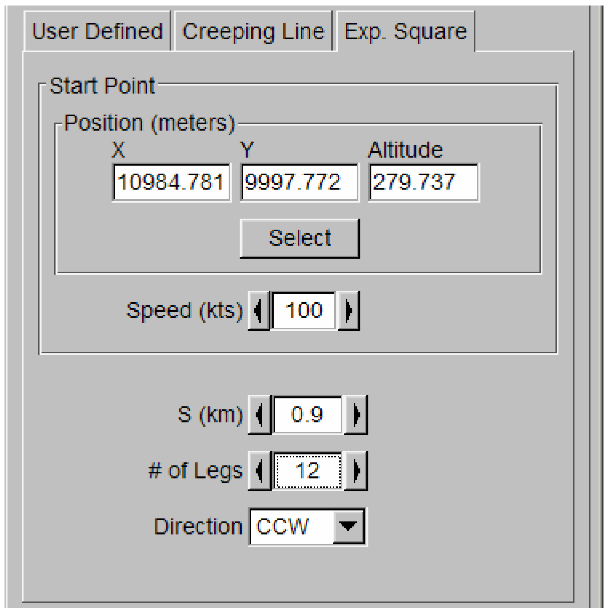


Figure 27. *Expanding Square Search Pattern*

j. Modifying the Expanding Square Flight Pattern

The Expanding Square flight pattern is created based on a number of parameters. Changing these parameters will affect the geographic area that the pattern will cover and the rate at which the area is covered. The following table describes the various parameters that affect the Expanding Square flight pattern and how to modify them. The Expanding Square manipulation interface is depicted in Figure 28.



X: 10984.781
Y: 9997.772
Altitude: 279.737
Speed (kts): 100
S (km): 0.9
# of Legs: 12
Direction: CCW

Figure 28. *Expanding Square Flight Pattern Manipulation Interface*

Table 15. *Expanding Square Parameters*

PARAMETER NAME	**DESCRIPTION**	**USE**
Position	Assigns the geographic location of the starting point for the flight pattern. The position is specified by an x, y, and Altitude value. These values are in meters.	The Start Point position may be selected interactively by clicking the **Select** button and then clicking on a location on the MMD. Alternatively, you may select and type discrete values into the **x**, **y** and **Altitude** fields and then hit <Enter> to apply the new value(s).
Speed	Defines the speed at which the simulated search aircraft will fly the flight pattern.	Click on the increment or decrement controls on the **Speed** spin box to increase or decrease the speed of the simulated search aircraft.
S	Represents the separation between the flight legs that make up the pattern. S is measured in kilometres.	Click on the increment or decrement controls on the **S** spin box to increase or decrease the leg separation. This value should be at least 0.1 km.
# of Legs	Assigns the number of flight legs that will make up the flight pattern.	Click on the increment or decrement controls on the **# of Legs** spin box to increase or decrease the number of legs that make up the pattern.
Direction	Dictates the direction of the first turn that is made in the pattern. The direction may be clockwise or counter-clockwise.	Select the desired direction from the **Direction** pick-list.

Figure 29 describes the Expanding Square flight pattern and the parameters used to generate it.

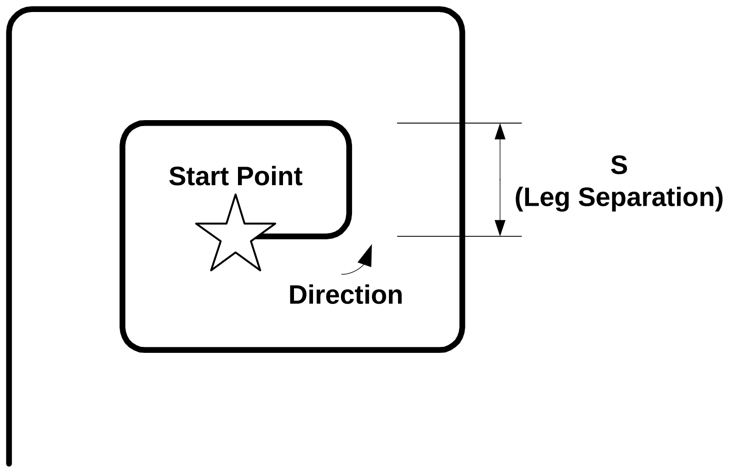


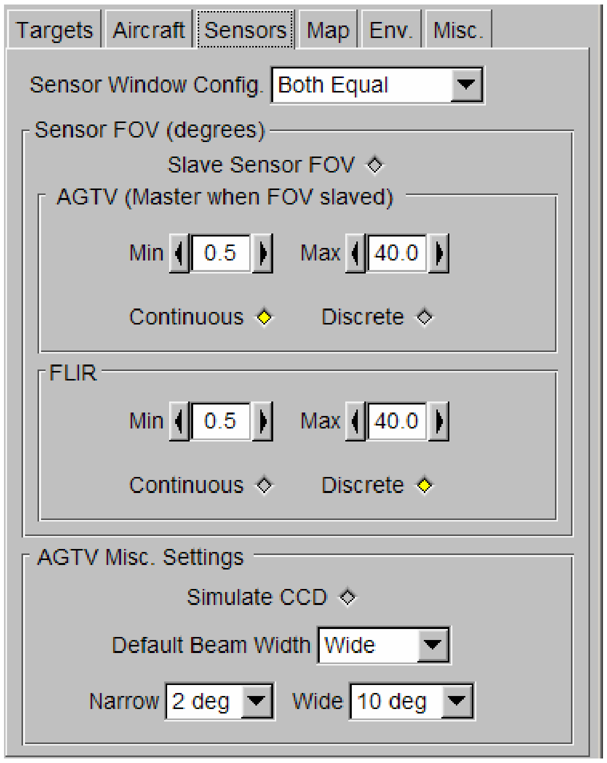
Figure 29. *Expanding Square Parameters*

k. Saving a Flight Plan

Whenever you make changes to your flight plan you will want to preserve those changes by saving the flight plan. To save the flight plan, select **File | Save Flight Plan** on the SGE window menu bar. Use the file browser to navigate to the desired directory click **OK** to save the flight plan. Alternatively you may wish to overwrite an existing flight plan file. In this case use the file browser to navigate to the directory that contains the file that you wish to overwrite and click **OK.**

5. Manipulating Sensors

At the core of the AIMSsim are two electro-optical sensors: an AGTV and a FLIR. The SGE allows you to control a number of sensor parameters. These parameters affect the way the simulated sensors will operate, as well as the way in which they will be presented in the HMI Prototype. The sensor manipulation interface is presented in Figure 30.



Sensor Window Config. Both Equal

Sensor FOV (degrees)

Slave Sensor FOV

AGTV (Master when FOV slaved)

Min 0.5 Max 40.0

[✓ Continuous] Discrete

FLIR

Min 0.5 Max 40.0

Continuous [✓ Discrete]

AGTV Misc. Settings

Simulate CCD

Default Beam Width Wide

Narrow 2 deg Wide 10 deg

Figure 30. *Sensor Manipulation Interface*

The following table describes the various sensor parameters and how to modify them. Field of view is referred to as the FOV.

Table 16. *Sensor Parameters*

PARAMETER NAME	DESCRIPTION	USE
Sensor Window Configuration	Controls the layout of the sensor video presentation in the HMI Prototype. Valid settings are “Both Equal”, “AGTV Primary”, “FLIR Primary”, “AGTV Only” AND “FLIR Only”. Currently, “Both Equal” uses two 640x480 screen areas for the sensor display. The only and Primary modes use a larger 4:3 section of the screen for the primary sensor (768x576).	Select the desired sensor window configuration from the **Sensor Window Configuration** pick-list.
Slave Sensor FOV	“Slaves” the FLIR sensor FOV to that defined for the AGTV. ie. the FOV’s for both are the same, only one control needed to zoom in/out.	Click on the **Slave Sensor FOV** radio button to toggle the FOV slave setting.
AGTV Min	Assigns the minimum allowable AGTV FOV that may be achieved in the HMI Prototype.	Click on the increment or decrement controls of the **Min** spin box to increase or decrease the minimum sensor FOV.
AGTV Max	Assigns the maximum allowable AGTV FOV that may be achieved in the HMI Prototype.	Click on the increment or decrement controls of the **Max** spin box to increase or decrease the maximum sensor FOV.

AGTV Continuous	Defines the zoom control for the AGTV (in the HMI Prototype) to be a continuous range between the AGTV Minimum FOV and the AGTV Maximum FOV.	Click on the **Continuous** radio button to toggle the FOV setting between “Continuous” and “Discrete”.
AGTV Discrete	Defines the zoom control for the AGTV (in the HMI Prototype) to be a discrete setting. This discrete setting may be selected at runtime by the HMI Prototype operator to be the AGTV Minimum FOV or the AGTV Maximum FOV.	Click on the **Discrete** radio button to toggle the FOV setting between “Continuous” and “Discrete”.
FLIR Min	Assigns the minimum allowable FLIR FOV that may be achieved in the HMI Prototype.	Click on the increment or decrement controls of the **Min** spin box to increase or decrease the minimum sensor FOV. This control has no effect when **Slave Sensor FOV** is selected.
FLIR Max	Assigns the maximum allowable FLIR FOV that may be achieved in the HMI Prototype.	Click on the increment or decrement controls of the **Max** spin box to increase or decrease the maximum sensor FOV. This control has no effect when **Slave Sensor FOV** is selected.
FLIR Continuous	Defines the zoom control for the FLIR (in the HMI Prototype) to be a continuous range between the FLIR Minimum FOV and the FLIR Maximum FOV.	Click on the **Continuous** radio button to toggle the FOV setting between “Continuous” and “Discrete”. This control has no effect when **Slave Sensor FOV** is selected.
FLIR Discrete	Defines the zoom control for the FLIR (in the HMI Prototype) to be a discrete setting. This discrete setting may be selected at runtime by the HMI Prototype operator to be the FLIR Minimum FOV or the FLIR Maximum FOV.	Click on the **Discrete** radio button to toggle the FOV setting between “Continuous” and “Discrete”. This control has no effect when **Slave Sensor FOV** is selected.
Simulate CCD	Enables/Disables the simulation of a CCD camera in place of the AGTV sensor simulation. It’s a camera with no laser illuminator (essentially a normal camera)	Click on the **Simulate CCD** radio button to toggle the CCD simulation setting between enabled (lit) and disabled (unlit).
Default Beam Width	Defines the default setting for the simulated laser illuminator beam width. The operator may override the default setting at runtime. Valid beam widths are “Wide” and “Narrow”.	Select the desired default beam width from the **Default Beam Width** selection list.
Narrow	Defines the size (in degrees) of the laser illuminator beam when the AGTV is operated in “Narrow” FOV mode. Valid settings are 2° and 5°.	Select the desired illuminator beam size from the **Narrow** selection list.
Wide	Defines the size (in degrees) of the laser illuminator beam when the AGTV is operated in “Wide” FOV mode. Valid settings are 10°, 15°, 25°,30° and 35°.	Select the desired illuminator beam size from the **Wide** selection list.

a. Saving the Sensor Configuration

Whenever you make changes to your sensor configuration you will want to preserve those changes by saving the scenario configuration settings. Unlike the target and flight plan parameters; the sensor configuration is saved together with the environmental settings and the Scenario Landscape definition. So keep in mind, when you are saving the sensor configuration, you are also saving those settings.

To save the sensor configuration, select **File | Save Config** on the SGE window menu bar. Use the file browser to navigate to the desired directory and click **OK** to save the configuration. Alternatively you may wish to overwrite an existing configuration file. In this case use the file browser to navigate to the directory that contains the file that you wish to overwrite and click **OK**

6. Manipulating the Map

The SGE allows you to control various HMI map options. The moving map is located in the upper right quadrant of the screen. When selecting a map mode this involves 8 options which include both 2D and 3D maps:

1) No map.
   The Map portion of the HMI is blank.
2) 2D Paper
   A correlated paper map image. This option requires that a papermap be available for the area in the terrain database. (Nerepis Only)
3) 2D Terrain
   A 2D moving map created using a bird's eye view of the terrain database. This is available for all databases.
4) 2D Shaded
   An elevation shaded representation of the terrain. This requires a special correlated database. (Nerepis Only)
5) Immersed
   3D view of the database from the sensor position. A blue “ghost-ball” is used to indicate the users field-of-view.
6) Immersed Shaded
   Same as Immersed, using the elevation shaded map.
7) Tethered
   3D view of the database from slightly above and behind the sensor position. A blue “ghost-ball” is used to indicate the users field-of-view. A 3D model is used to represent the search aircraft.
8) Tethered Shaded
   Same as Tethered, using the elevation shaded map.

Another option in manipulating the map allows for adjusting parameters. A scale can be selected from a range of 1000 to 250 000. A map scale will appear on the 2D map views for certain values. The ratios of 1: 1000, 10000, 25000, 50000, 100 000, 125 000, 200 000, and 250 000 will have a scale shown. Furthermore, 3 options are available for map orientation: 1) North Up (map is always oriented North regardless of the flight), 2) Aircraft Up (map orients according to aircraft position during flight), and 3) Camera Up (map orients according to camera position).

There are 4 options available for the aircraft symbol on the moving map: 1) Rotary Wing (simple helicopter-like icon), 2) Fixed Wing (simple aircraft-like icon), 3) Pointer (simple circle with an indicator, which points in the direction the search), and 4) There is also an option for no icon.

This section also provides a function for colour or greyscale. Search history can be enabled (selection of function on/function off). The search history marks a blue trail on the map, of the user's search. A Marking Function is also an option that can be enabled (selection of function on/function off). The marking function will show a numbered box on the map in the position of the designation when a 2D map is displayed in the MMD. Designations are number in chronological order. A white dot appears on the map in the position of the designation when a 3D map is displayed in the MMD.

7. Manipulating the Environment

The SGE allows you to control various simulated environmental parameters. These parameters will impact on the effectiveness of the simulated sensors in the HMI Prototype.

The following table describes the various environment parameters and how to modify them.

Table 17. *Environment Parameters*

PARAMETER NAME	DESCRIPTION	USE
Time of Day (TOD)	Controls the amount of ambient light illuminating the AGTV. Time of Day is measured on a scale from 0.0 to 1.0. A value of 0.0 represents complete darkness while a value of 1.0 represents maximum illumination.	Select the desired TOD setting by positioning the **Time of Day** slider. Alternatively, you may increment or decrement the TOD via the **Time of Day** spin box.
Visibility	Defines the visibility setting for the experimental scenario. Valid selections are "Clear" and "Degraded".	Select the desired visibility setting via the **Visibility** pick-list.
AGTV Effect	Determines the amount of degradation in the AGTV sensor. Selecting the setting "1" will result in complete degradation, selecting the setting "0" results in no degradation. (For example, the effect introduces noise.)	Click on the increment or decrement controls of the **AGTV Effect** spin box to increase or decrease the amount of degradation of the simulated AGTV sensor.
FLIR Effect	Determines the amount of degradation in the FLIR sensor. Selecting the setting "1" will result in complete degradation, selecting the setting "0" results in no degradation. (For example, the effect introduces noise.)	Click on the increment or decrement controls of the **FLIR Effect** spin box to increase or decrease the weather effectiveness of the simulated FLIR sensor.

a. Description of the Degradation Effect

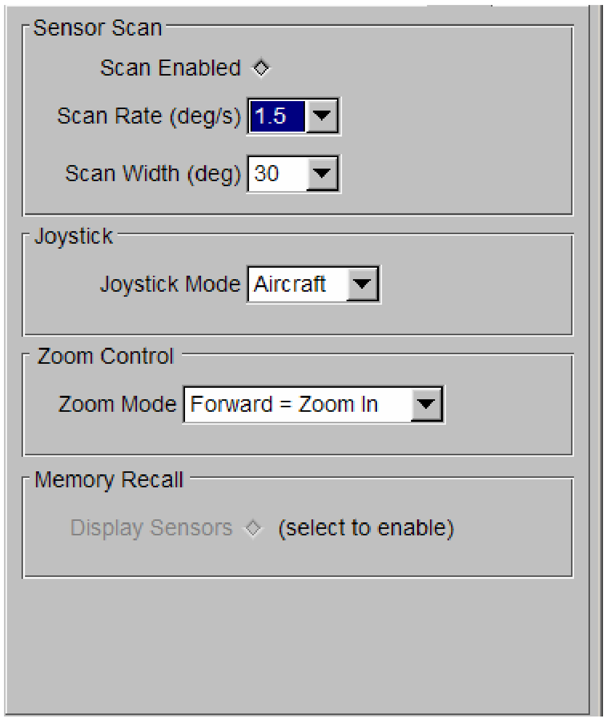
Computer images often appear unrealistically sharp and well defined. The degradation effect decreases this sharpness or the definition of these images through the addition of a transparent-noise-screen. A percentage of degradation can be selected on a scale of 0 to 1, which will remain consistent throughout the scene. The selection of zero results in no degradation, while 1 is complete degradation. Degradation is used to better simulate real world conditions.

b. *Saving the Environment Configuration*

The environment configuration is saved together with the sensor configuration and the Scenario Landscape information. For detailed information on saving the environment configuration, please refer to Section a - Saving the Sensor Configuration.

8. Manipulating Additional Scenario Settings

The SGE allows to you control additional scenario settings. These miscellaneous settings control the default configuration of the HMI Prototype display and of the HMI control hardware. The miscellaneous settings manipulation interface is presented in Figure.



Sensor Scan
Scan Enabled ◇
Scan Rate (deg/s) 1.5
Scan Width (deg) 30
Joystick
Joystick Mode Aircraft
Zoom Control
Zoom Mode Forward = Zoom In
Memory Recall
Display Sensors ◇ (select to enable)

Figure 31. *Miscellaneous Settings Manipulation Interface*

The following table describes the various miscellaneous parameters and how to modify them.

Table 18. *Miscellaneous Parameters*

PARAMETER NAME	DESCRIPTION	USE
Scan Enabled	Controls the default state (enabled/disabled) of the automatic sensor scan function in the HMI Prototype. When the scan is enabled, the terrain is automatically scanned with no movement of the joystick.	Select the default state of the scan function: enabled (lit) or disabled (unlit) by clicking on the **Scan Enabled** radio button.

Scan Rate	Controls the default scan rate of the automatic sensor scan function in the HMI Prototype. Valid scan rates are “1.5 deg/s”, “3 deg/s” and “6.0 deg/s”.	Select the desired scan rate setting from the **Scan Rate** pick-list.
Scan Width	Controls the width of the default scan sweep by the automatic sensor scan function in the HMI Prototype. Valid scan widths are “30 deg”, “60 deg” and “90 deg”.	Select the desired scan width setting from the **Scan Width** pick-list.
Joystick Mode	Controls the default manipulation mode of the HMI control hardware joystick. Valid manipulation modes are “Aircraft” and “Cursor”. When in aircraft mode, pushing forward on the joystick results in the sensor turret pitching down and pulling back on the joystick results in the sensor turret pitching up. When in cursor mode, the pitch control relationship is reversed: pushing forward on the joystick results in the sensor turret pitching up and pulling back on the joystick results in the sensor turret pitching down.	Select the desired joystick mode setting from the **Joystick Mode** pick-list.
Zoom Control Mode	Controls the default manipulation mode of the HMI control hardware zoom levers. Valid manipulation modes are “Forward = Zoom In” and “Forward = Zoom Out”. When in “Forward = Zoom In” mode, pushing forward on the zoom lever results in the sensor zooming in and pulling back on the zoom lever results in the sensor zooming out. When in “Forward = Zoom Out” mode, the zoom control relationship is reversed: pushing forward on the zoom lever results in the sensor zooming out and pulling back on the zoom lever results in the sensor zooming in.	Select the desired zoom control mode setting from the **Zoom Control Mode** pick-list.
Display Sensors	Allows the Memory Recall capability to be enabled. In this mode, no sensor imagery will be displayed.	If this box is not checked, the prototype will run in Memory Recall mode.

9. Altitude Profile Display

The Altitude Profile Display provides a graphical representation of the simulated search aircraft altitude profile and terrain elevation profile with respect to the scenario timeline. The data points that create the aircraft altitude profile are a result of the waypoints defined in the flight plan. As such, the numbers represented on the altitude profile correspond to waypoint numbers in the flight plan. The aircraft altitude profile is drawn in blue and the terrain elevation profile is drawn in green. The altitude profile display is presented in Figure 29.

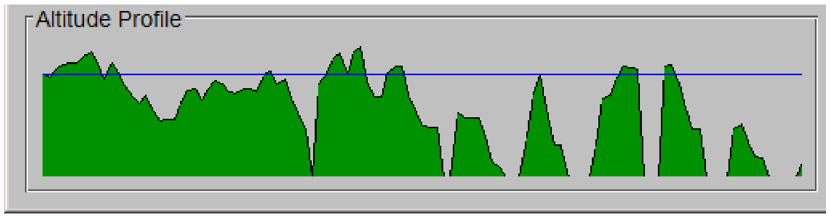


Figure 29. *Altitude Profile Display*

10. The Scenario Summary Area

The Scenario Summary Area provides summary information about the AIMSsim experimental scenario that you have created. The summary information includes an estimated time for executing the scenario as well as information about the total number of targets and non-targets defined in the scenario.

11. Managing Your Projects

A project is a logical means of grouping target information, flight plan information and HMI configuration information (including sensor, environment and scenario landscape information) to form a complete experimental scenario.

a. Saving Your Project

Once you have defined these elements of scenario information, you may preserve the logical grouping by saving your project. To save a project, select **File | Save Project** on the SGE window menu bar. Use the file browser to navigate to the desired directory and click **OK** to save the project. This will create four files in the selected directory which will store all of the information for the current project (i.e. ‘project.xml’, ‘flightplan.xml’, ‘config.xml’ and ‘targets.xml). Alternatively you may wish to overwrite an existing project. In this case use the file browser to navigate to the directory that contains the project files that you wish to overwrite and click **OK.**

b. Loading an Existing Project

To load an existing project, select **File | Open Project** on the SGE window menu bar. Use the file browser to navigate to the directory than contains the existing project files and select the ‘project.xml’ file - click **OK**. The SGE will load the project parameters contained in these files.

H.6 Viewing a Specific Target Object

In previous sections explaining the use of the AIMSsim SGE, mention was made of a Target view mode. This mode is toggled from within the SGE to view the placement and orientation of target objects in 3D space with respect to the surrounding terrain.

H.7 Scenario Definition Files

The preferred method of creating Scenario Definition files is via the AIMSsim SGE. This is particularly true with regard to the current version of the SGE as the file format has been changed from plain ASCII text files to XML encoded files. The content of these files is still viewable using a simple text editor. Since two of the files that are generated by the SGE (flightplan.xml and targets.xml) can be read natively by the AIMSsim HMI, only the SGE configuration file (config.xml) will be discussed here. In addition, a description of the target mapping file will also be presented. It is not recommended that any of these files be edited outside of the SGE as this may cause improper functioning.

1. Configuration File

The Configuration file contains information about the Scenario Landscape, the configuration of the simulated sensors and the configuration of the HMI Prototype. It is divided in five main sections each delimited by an XML tag set. An example of the SGE configuration file is given below.

```
<config>
    <terrain>c:\AimsDB/terrains/Nerepis/ELVISS_ne.ive</terrain>
    <environment>
            <tod>1</tod>
            <vis>0</vis>
            <agtv_effect>1</agtv_effect>
            <flir_effect>2</flir_effect>
    </environment>
    <sensor>
            <win_cfg>0</win_cfg>
            <slaved>0</slaved>
            <agtv_fov_min>0.5</agtv_fov_min>
            <agtv_fov_max>40</agtv_fov_max>
            <agtv_mode>0</agtv_mode>
            <agtv_beam_width>0</agtv_beam_width>
            <agtv_sim_ccd>0</agtv_sim_ccd>
            <agtv_narrow_size>0</agtv_narrow_size>
            <agtv_wide_size>0</agtv_wide_size>
            <flir_fov_min>0.5</flir_fov_min>
            <flir_fov_max>40</flir_fov_max>
            <flir_mode>1</flir_mode>
    </sensor>
    <misc>
            <scan_rate>0</scan_rate>
            <scan_width>0</scan_width>
            <scan_enable>0</scan_enable>
            <joy_mode>0</joy_mode>
```

```
                <zoom_mode>0</zoom_mode>
        </misc>
        <map>
                <mode>0</mode>
                <scale>50000</scale>
                <orient>0</orient>
                <symbol>0</symbol>
                <colour>0</colour>
                <search_history>0</search_history>
                <mark_function>0</mark_function>
                <recall_sensor>0</recall_sensor>
                <fov>60</fov>
                <slave_fov>0</slave_fov>
        </map>
</config>
```

The entire configuration file is enclosed within the <config>, </config> tag set. The five major sections fall within this global set. The first major section of the configuration file occurs within the <terrain> tag. The value contained in this tag represents an absolute path to the location of the terrain file being used with the current scenario. The four other sections (sensor, environment, map and misc) are delimited by XML tags similarly named. These four sections correspond to the latter four tabs that occur in the SGE GUI. Each of these sections contains properties that can be both viewed and set from the appropriate tab in the SGE GUI. A description of these properties now follows.

Table 19. *Configuration File Specification*

XML TAGS	DESCRIPTION	VALUE
<terrain>	The absolute file path to the terrain database file.	A valid file path.
<environment>	The second major section of the configuration file which contains all environment settings.	N/A
<tod>,</tod>	The Time of Day (TOD) to be represented in the HMI Prototype.	0.0 – 1.0
<vis>,</vis>	The weather condition to be represented in the HMI Prototype.	0 = Clear 1 = Degraded
<agtv_effect>, </agtv_effect>	The capability of the simulated AGTV sensor to penetrate the fog.	0.0 – 1.0
<flir_effect>, </flir_effect>	The capability of the simulated FLIR sensor to penetrate the fog.	0.0 – 1.0

<sensor>	The third major section of the configuration file which contains all of the sensor settings.	N/A
<win_cfg>, </win_cfg>	An integer value that controls the layout of the sensor video presentation in the HMI Prototype.	0 = Both Equal 1 = AGTV Primary 2 = FLIR Primary 3 = AGTV Only 4 = FLIR Only
<slaved>, </slaved>	A flag that, when set, “Slaves” the FLIR sensor FOV to that defined for the AGTV.	0 = not slaved 1 = slaved
<agtv_fov_min>, </agtv_fov_min>	A single precision floating point value that assigns the minimum allowable AGTV FOV that may be achieved in the HMI Prototype. Measured in degrees.	0.5 – 40.0
<agtv_fov_max>, </agtv_fov_max>	A single precision floating point value that assigns the maximum allowable AGTV FOV that may be achieved in the HMI Prototype. Measured in degrees.	0.5 – 40.0
<agtv_mode>, </agtv_mode>	An integer value that controls the zoom control for the AGTV.	0 = Continuous 1 = Discrete
<agtv_beam_width>, </agtv_beam_width>	An integer value that defines the default width of the illuminator beam.	0 = Wide 1 = Narrow
<agtv_sim_ccd>, </agtv_sim_ccd>	An integer value that defines the state of CCD simulation.	0 = Do not simulate CCD 1 = Simulate CCD
<agtv_narrow_size>, </agtv_narrow_size>	An integer value that defines the size (in degrees) of the AGTV laser illuminator beam when operating in “Narrow” FOV mode.	0 = 2 deg 1 = 5 deg
<agtv_wide_size>, </agtv_wide_size>	An integer value that defines the size (in degrees) of the AGTV laser illuminator beam when operating in “Wide” FOV mode.	0 = 10 deg 1 = 15 deg 2 = 20 deg 3 = 25 deg 4 = 35 deg
<flir_fov_min>, </flir_fov_min>	A single precision floating point value that assigns the minimum allowable FLIR FOV that may be achieved in the HMI Prototype. Measured in degrees.	0.5 – 40.0
<flir_fov_max>, </flir_fov_max>	A single precision floating point value that assigns the maximum allowable FLIR FOV that may be achieved in the HMI Prototype. Measured in degrees.	0.5 – 40.0
<flir_mode>, </flir_mode>	An integer value that controls the zoom control for the FLIR.	0 = Continuous 1 = Discrete
<agtv_sim_ccd>, </agtv_sim_ccd>	Enables/Disables the simulation of a CCD camera in place of the AGTV sensor simulation.	Select to enable (0=inactive, 1=active)

<map>	The fourth major section of the configuration file which contains all properties pertaining to the map.	N/A
<mode>, </mode>	Defines the map (upper right quadrant) for the scenario. Valid map selections are “No Map”, “2D Paper”, “2D Terrain”, “2D Shaded”, “Immersed”, “Immersed Shaded”, “Tethered” and “Tethered Shaded”	Select the desired map mode setting from the Map Mode pick-list
<scale>, </scale>	Controls the default orientation setting for the HMI Prototype MMD. Valid map scales range from 1:1 to 1:250,000.	Select the desired map scale using either scale bar selection, or manual entry.
<orient>, </orient>	Defines the default orientation setting for the HMI prototype MMD. Valid map orientations are “North Up”, “Aircraft Up”, and “Camera Up”.	Select the desired map orientation setting from the Map Orientation pick-list
<symbol>, </symbol>	Controls the default symbol used to represent the position and orientation of the simulated search aircraft for the HMI Prototype MMD. Valid aircraft symbols are “Rotary Wing”, “Fixed Wing”, “Pointer” and “No Icon”.	Select the desired aircraft symbol setting from the Aircraft Symbol pick-list
<colour>, </colour>	Defines the colour of the terrain as either colour or greyscale.	Select the desired colour setting from the Colour Mode pick- list.
<search_history>, </search_history>	Enables a function where blue markings reveal past, or most recent search on MMD.	Select to enable (0=inactive, 1=active)
<mark_function>, </mark_function>	Enables the designated function.	Select to enable (0=inactive, 1=active)
<fov>, </fov>	Represents the Map field of view.	Select a value from 1 – 120 deg.
<slave_fov>, </slave_fov>	Enabled only when MMD is “Immersed” or “Tethered”. It is similar to having an additional sensor, which allows for zoom.	Select to enable (0=inactive, 1=active)
<misc>	The fifth section of the configuration file which contains miscellaneous settings.	N/A
<scan_enable>, </scan_enable>	An integer value that controls the default scan state setting.	Select to enable (0=inactive, 1=active)
<scan_rate>, </scan_rate>	An integer value that controls the default scan rate setting.	0 = 1.5 deg/s 1 = 3.0 deg/s 2 = 6.0 deg/s
<scan_width>, </scan_width>	An integer value that controls the default scan sweep width setting.	0 = 30 deg 1 = 60 deg 2 = 90 deg
<joy_mode>, </joy_mode>	An integer value that controls the default joystick mode setting.	0 = aircraft mode 1 = cursor mode

<zoom_mode>, </zoom_mode>	An integer value that controls the default zoom mode setting.	0 = Forward-Zoom In 1 = Forward-Zoom Out
<recall_sensor>, </recall_sensor>	Permission for sensors to be activated (not a default function).	Select to enable (0=inactive, 1=active)

2. Target Mapping File

The Target Mapping file relates a “descriptive” name to a 3D model file name. This approach was used in order to provide a meaningful name to the SGE user, while maintaining the capability to represent an appropriate 3D model in the HMI Prototype.

The Target Mapping file contains two columns of alpha-numeric data. The first column contains the descriptive name of the object, while the second column contains the file name of the 3D model.

Both the SGE and the HMI Prototype read the contents of the Target Mapping file at runtime. As such, you may add additional targets to the Target Mapping file by following the simple two-column format. The target *Type* field present in the Target Definition file utilizes a numerical index into the Target Mapping file. For this reason it is recommended that any new entries be appended to the *end* of the file so as not to invalidate any scenarios that were developed prior to your modifications.

The current target mapping file in the database is called `target_map.txt`.

Bibliography

1. Gamble, M. (2001). *ELVISS Software Prototype - System Manual*. The HFE Group. Defence Research and Development Canada - Toronto Contract Report: DRDC Toronto CR 2001-029.

2. Gamble, Murray G. (November 10, 2000). *Proposal to Modify ELVISS Prototype Software. Volume One - Technical and Management Proposal*. The HFE Group.

3. Neal, B. (April 28, 1999). *ELVISS Human Engineering Design Approach Document – Operator*. Canadian Marconi Company, Human Factors Engineering Aerospace Division. Defence Civil Institute of Environmental Medicine Contract Report Number: DCIEM-CR-1999-078CMC-1000-1182.

4. Neider, Jackie. (1993). *OpenGL Programming Guide*. Addison Wesley.

5. Schoenborn, O. (2006a). *AIMSsim version 2.2.1 User Manual*. CAE Professional Services, Ottawa, Ontario. Defence Research and Development Canada - Atlantic: DRDC Atlantic CR 2006-280.

6. Schoenborn, O. (2006b). *AIMSsim version 2.2.1 System Manual*. CAE Professional Services, Ottawa, Ontario. Defence Research and Development Canada - Atlantic: DRDC Atlantic CR 2006-270.

7. Schoenborn, O. (2006c). *Herc SAR Task 106: AIMS Feature Development Final Report*. CAE Professional Services, Ottawa, Ontario. Defence Research and Development Canada - Atlantic: DRDC Atlantic CR 2006-278.

List of symbols/abbreviations/acronyms/initialisms

AGTV	Active Gated TV
AIMS	Advanced Integrated Multi-sensor Surveillance
AIMSsim	AIMS simulator
ALBEDOS	Airborne Laser-Based Enhanced Detection and Observation System
ASCII	American Standard Code for Information Interchange
ATD	Automatic Target Detection
CCD	Charge-Coupled Device
DND	Department of National Defence
DREV	Defence Research Establishment Valcartier
DRDC	Defense Research Development Canada
ELVISS	Enhanced Low-Light Level Visible and Infrared Surveillance System
EO	Electro-Optic
FLIR	Forward Looking Infrared
FLTK	Fast Light Tool Kit
FOV	Field of View
FSM	Finite State Machine
GUI	Graphical User Interface
HMI	Human Machine Interface
JPEG	Joint Pictures Expert Group
LOS	Line of sight
IR	Infrared
MMD	Moving Map Display
N/A	Not Applicable
PC	Personal Computer
POV	Point of View
SAR	Search and Rescue
SGE	Scenario Generation Environment
SGI	Silicon Graphics, Inc.

THIR	Thermal Infra –Red (imager)
TOD	Time of Day
USB	Universal Serial Bus
VAPS	Virtual Applications Prototyping System
VPI	Virtual Prototypes, Inc.
XML	Extensible Markup Language
Wx	Weather

This page intentionally left blank

Distribution list

Document No.: DRDC Atlantic CR 2007-300

LIST PART 1: Internal Distribution by Centre:

5	DRDC Atlantic Library
2	DRDC Atlantic Scientific Authority
7	TOTAL LIST PART 1

LIST PART 2: External Distribution by DRDKIM

1	Jocelyn Keillor DRDC Toronto PO Box 2000 Toronto, Ontario M3M 3B9 Canada
1	DRDKIM
2	TOTAL LIST PART 2

9 TOTAL COPIES REQUIRED

This page intentionally left blank

DOCUMENT CONTROL DATA
(Security classification of the title, body of abstract and indexing annotation must be entered when the overall document is classified)

1. ORIGINATOR (The name and address of the organization preparing the document, Organizations for whom the document was prepared, e.g. Centre sponsoring a contractor's document, or tasking agency, are entered in section 8.) Publishing: DRDC Atlantic Performing: CAE Professional Services, 1135 Innovation Dr., Ottawa, ON, K2K 3G7 Monitoring: Contracting:	2. SECURITY CLASSIFICATION (Overall security classification of the document including special warning terms if applicable.) UNCLASSIFIED

3. TITLE (The complete document title as indicated on the title page. Its classification is indicated by the appropriate abbreviation (S, C, R, or U) in parenthesis at the end of the title)
AIMSsim version 2.3.4 − User Manaul (U)

4. AUTHORS (First name, middle initial and last name. If military, show rank, e.g. Maj. John E. Doe.)
O. Schoenbom; P. Lachance; N. Bahramifarid

5. DATE OF PUBLICATION (Month and year of publication of document.) January 2008	6a NO. OF PAGES (Total containing information, including Annexes, Appendices, etc.) 116	6b. NO. OF REFS (Total cited in document.) 4

7. DESCRIPTIVE NOTES (The category of the document, e.g. technical document, technical note or memorandum. If appropriate, enter the type of document, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)
Contract Report User manual for a multi−sensor simulator research platform that includes sensor interface and controls and airborne scene. Associated documents: DRDC Atlantic CR 2007-301 − System Manual, DRDC Atlantic CR 2007-302 − AIMSsim Final Report

8. SPONSORING ACTIVITY (The names of the department project office or laboratory sponsoring the research and development − include address.)
Sponsoring:
Tasking: DRDC Atlantic
DRDC Valcartier

9a. PROJECT OR GRANT NO. (If appropriate, the applicable research and development project or grant under which the document was written. Please specify whether project or grant.) 13dx	9b. CONTRACT NO. (If appropriate, the applicable number under which the document was written.) W7711−047904/TOR/001
10a. ORIGINATOR'S DOCUMENT NUMBER (The official document number by which the document is identified by the originating activity. This number must be unique to this document) DRDC Atlantic CR 2007−300	10b. OTHER DOCUMENT NO(s). (Any other numbers under which may be assigned this document either by the originator or by the sponsor.)

11. DOCUMENT AVAILABILITY (Any limitations on the dissemination of the document, other than those imposed by security classification.)
Unlimited distribution

12. DOCUMENT ANNOUNCEMENT (Any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11), However, when further distribution (beyond the audience specified in (11) is possible, a wider announcement audience may be selected.))
Unlimited announcement

DOCUMENT CONTROL DATA

(Security classification of the title, body of abstract and indexing annotation must be entered when the overall document is classified)

13. ABSTRACT (A brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), (R), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual.)

(U) This user manual provides an overview of the software functionality developed to support the empirical investigation of a simulated user interface for an Advanced Integrated Multi−sensor Surveillance (AIMS) system (formerly known as the Enhanced Low−Light Level Visible and Infrared Surveillance System – ELVISS). The AIMS system is an electro−optical imaging system being developed by Defence Research and Development Canada (DRDC) – Valcartier to enhance the capability of search and rescue (SAR) crews to operate effectively at night and in degraded weather conditions. In order to ensure that a SAR operator would be able to use the system effectively and with a minimal amount of training, a prototype human−machine interface (HMI) was developed to evaluate design concepts. The latest development phase added sensor controller options and a configurable display interface for evaluating interface design concepts.

(U) Le présent manuel de l’utilisateur donne un aperçu des fonctions du logiciel élaboré pour prendre en charge l’examen empirique d’une interface utilisateur simulée d’un système perfectionné de surveillance multi−capteurs intégré (AIMS) (anciennement appelé Système perfectionné de surveillance à intensification de lumière visible et à infrarouge ou ELVISS). Le système AIMS est un système d’imagerie électro−optique en cours de développement à Recherche et développement pour la défense Canada (RDDC) – Valcartier pour améliorer la capacité des équipages de recherche et de sauvetage (SAR) à travailler avec efficacité la nuit et par intempérie. Afin de garantir qu’un opérateur du service SAR sera en mesure d’utiliser le système efficacement et moyennant une formation minimale, un prototype d’interface humain−machine (IHM) a été développé pour évaluer les principes de conception. La dernière phase de développement consistait à ajouter des options de contrôleur des capteurs et une interface d’affichage configurable pour évaluer les principes de conception d’interface.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (Technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus, e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

(U) simulation; airborne; surveillance; search and rescue; electro−optical sensor; operator−machine interface