# FlightViz Simulation Tool 2006
# User & Administrator's Manual

## Michigan State University

In Collaboration with

## The Boeing Company


**Boeing Representatives**
Jayson Vincent
Lanya da Silva
Kim Monteith

**Michigan State University Capstone Students**
Alan Antonuk: antonuka@msu.edu
Chris Tobey: tobeychr@msu.edu
Eric Vogel: vogeler1@msu.edu
Chae Williams: will1157@msu.edu

# Table of Contents

# 1. FlightViz Overview

FlightViz 2006 is a real-time, networked flight simulation viewer. The viewer displays airplanes and other objects described by the Distributed Interactive Simulation (DIS) protocol packets received over TCP/IP. FlightViz includes accurate terrain data for Illinois (IL), Michigan (MI), Missouri (MO), and California (CA). FlightViz 2006 also integrates 3D models of an F-18 A/B and a MiG29 Fulcrum, with plug-in support for additional models.

# 2. System Requirements

**Minimum Tested System Requirements**

Intel® Pentium® IV 2.0 GHz
1 GB RAM
1 MB of Hard Drive Space for the program
CA, IL, MI, MO (~10 GB additional Hard Drive Space)
128 MB NVIDIA® GeForce FX 5200 or equivalent
DirectX® 9.0c (with Managed Extensions)
.NET Framework 2.0

# 3. User Manual

## 3.1. TestServer

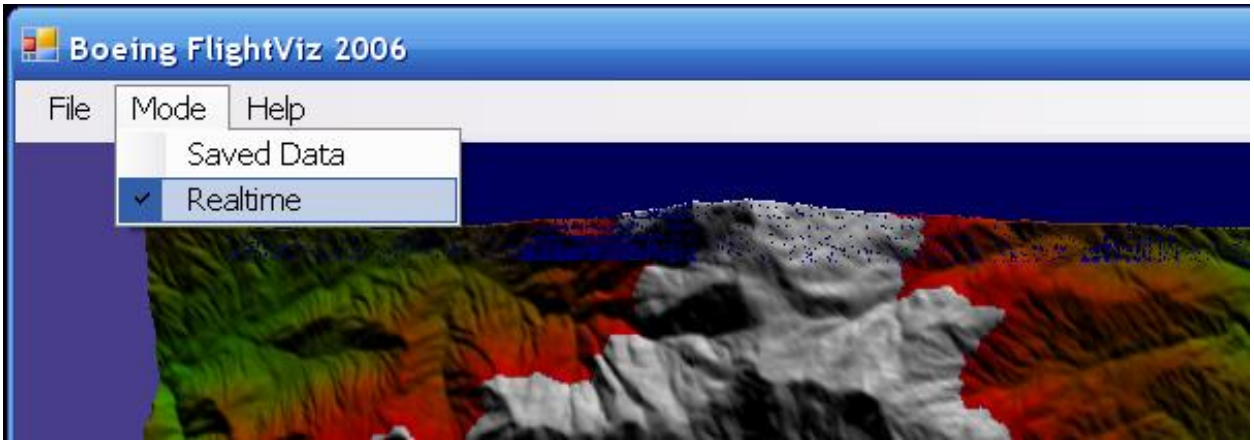
Launch FlightViz, and then launch the TestServer. The TestServer will then start sending DIS PDUs to FlightViz according to its preprogrammed flight path.

## 3.2. FlightViz

### 3.2.1. Running Modes

#### 3.2.1.1. Changing Run Modes

To change the run mode click on Mode in the top Menu. A drop down will display the two types of run modes: saved data and real-time. Select the desired run mode.

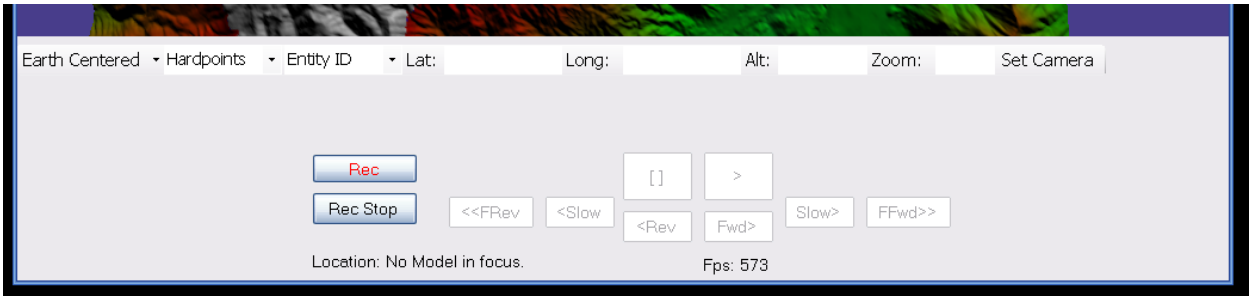


### 3.2.1.2. Real-time

This mode is used to view a DIS simulation session currently being received over a network.

#### *3.2.1.2.1. Recorder Controls*

The Rec button is in place for future use to record an active simulation session. The Rec Stop button is in place to terminate the recording of an active simulation session.



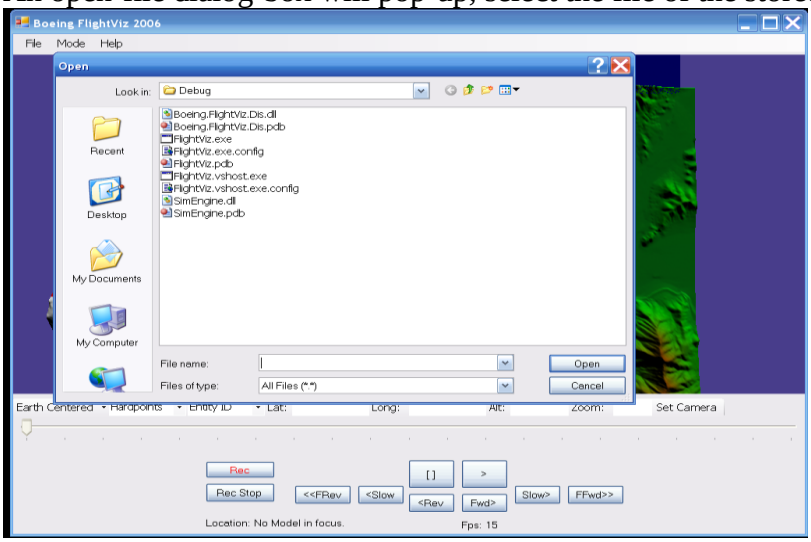

### 3.2.1.3. Saved Data

This mode is used to view a stored simulation session.

#### *3.2.1.3.1. Open Database*

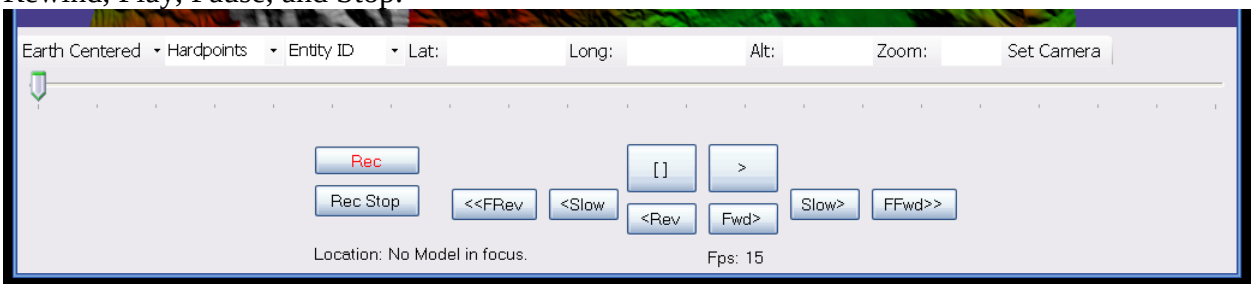
To open a simulation session from a database, click on File then Open Database from the drop down menu.


An open-file dialog-box will pop-up, select the file of the stored simulation session to open.



#### *3.2.1.3.2. Playback Controls*

The buttons and slide bar are used to play through a recorded session. They are in place for future use. They include: Fast Forward, Fast Rewind, Forward, Rewind, Slow Forward, Slow Rewind, Play, Pause, and Stop.

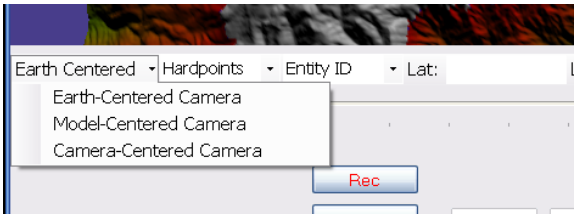



## 3.2.2. Cameras

FlightViz provides three camera types: an Earth-centered camera, a model-centric camera and a camera-centric camera.

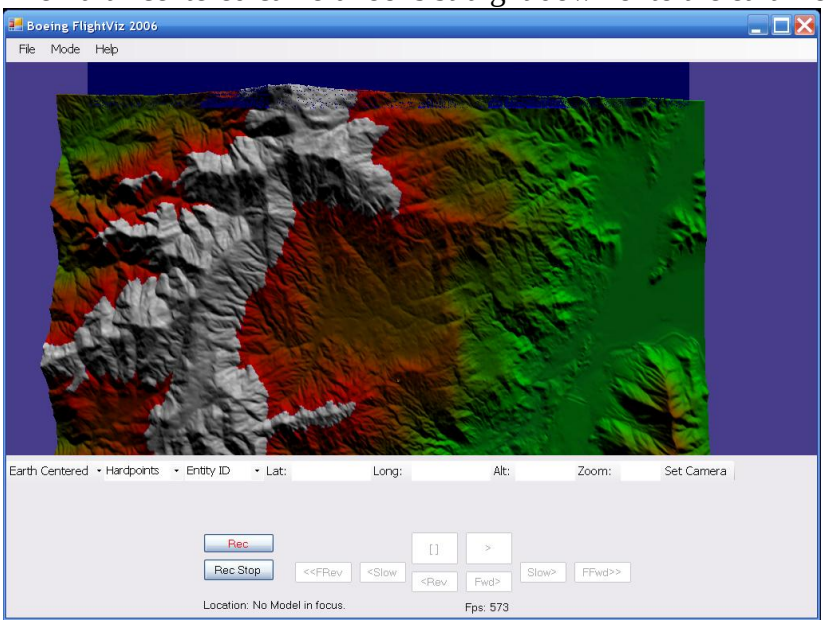
### 3.2.2.1. Changing Camera Type

The camera type can be changed by clicking on the camera type drop down bar and clicking on the desired camera type.



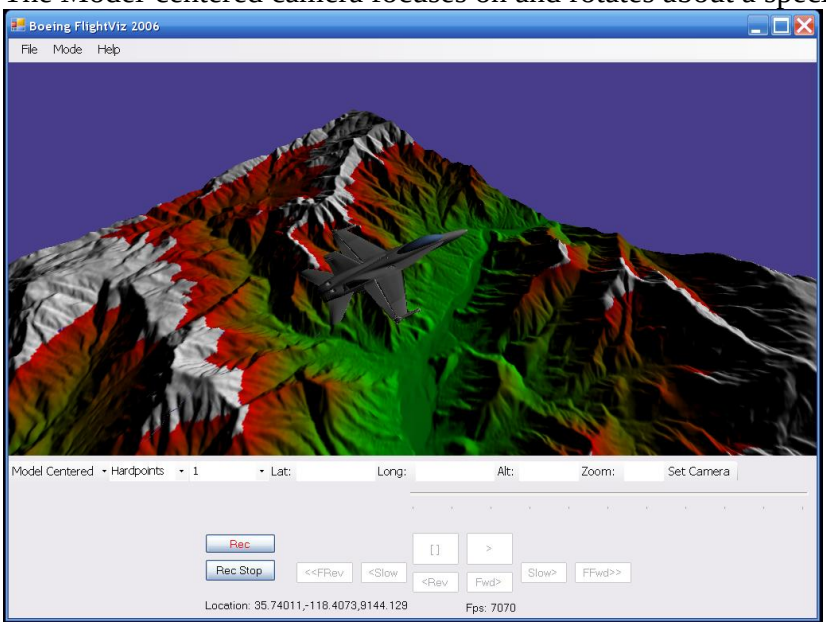

### 3.2.2.2. Earth-centered

The Earth-centered camera looks straight down onto the earth form a fixed location.



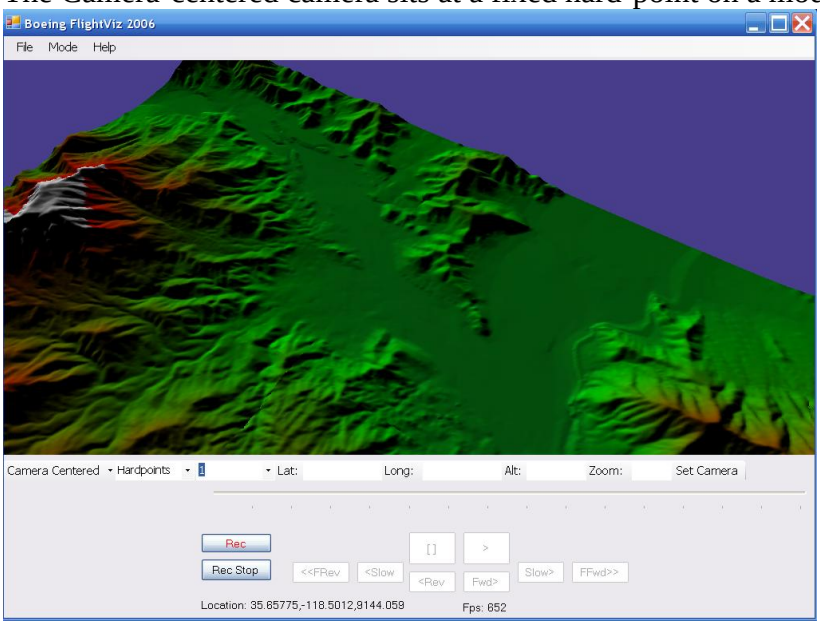
### 3.2.2.3. Model-centered

The Model-centered camera focuses on and rotates about a specified model.



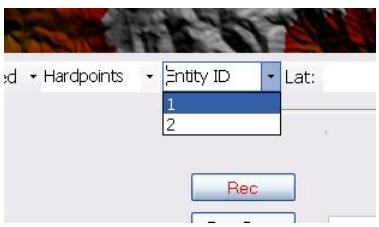

### 3.2.2.4. Camera-centered

The Camera-centered camera sits at a fixed hard-point on a model and looks around from there.



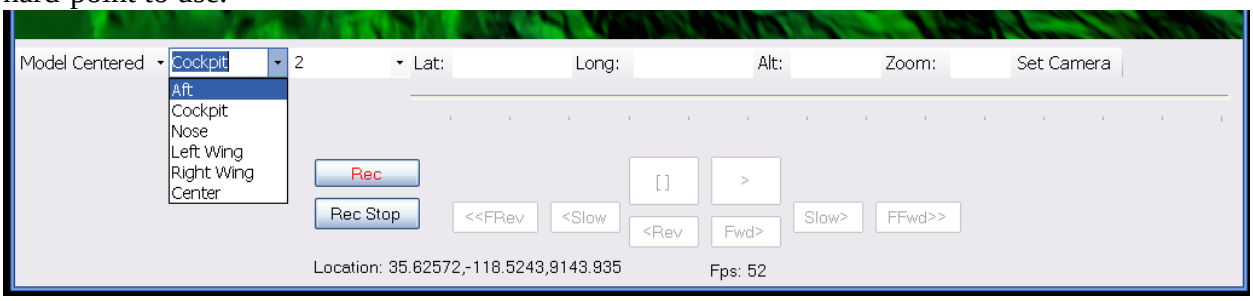

### 3.2.2.5. Selecting Focus Entity

To select the Focus Entity click on the Entity ID drop down bar and select the number of the entity to focus on. This number is the same identifier found in an Entity ID record of the DIS Framework. Once the number is clicked the camera-centric and model-centered cameras will follow that entity.



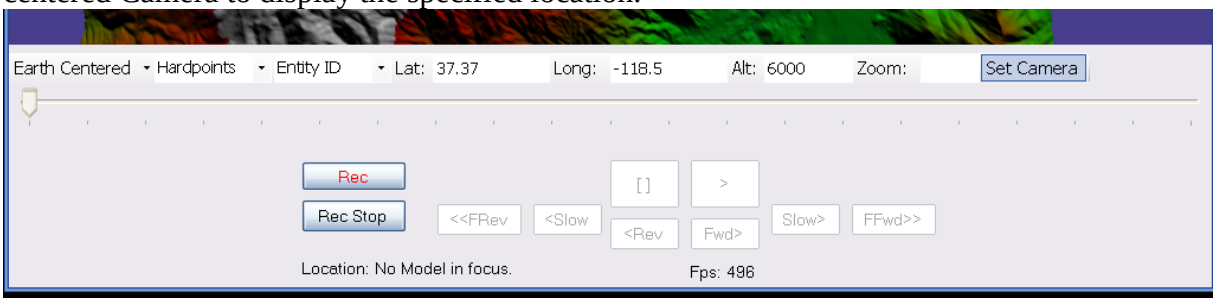
### 3.2.2.6. Selecting Entity Hard-Points

To select the entity hard-point to use click on the Hardpoints drop down bar and select which hard-point to use.




### 3.2.2.7. Changing camera position

To change the camera position type in the latitude in the Lat box, the longitude in the Long box, and the Altitude in the Alt box and click on the Set Camera button. This view uses an Earth-centered Camera to display the specified location.




### 3.2.2.8. Rotating the Camera

The keyboard can be used to rotate the model-centered and camera-centric cameras. The “I” and “K” keys will rotate the camera around the vertical axis; the “J” and “L” keys rotate the camera around the horizontal axis.

## 3.2.3. Models

FlightViz is provided with an F-18 A/B/ and a Mig29 model. FlightViz provides support for adding additional models through the use of model plug-ins. To add an additional model to FlightViz copy the DLL for the model along with the model mesh file and texture files into the "Models" directory. For details on creating a Model Plug-in, refer to the Administrator's Manual.

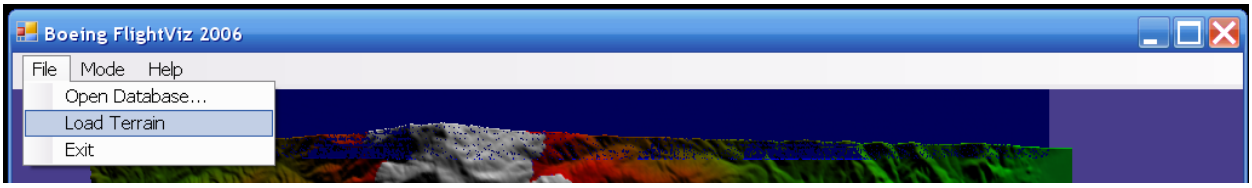
## 3.2.4. Terrain

### 3.2.4.1. Adding more terrain

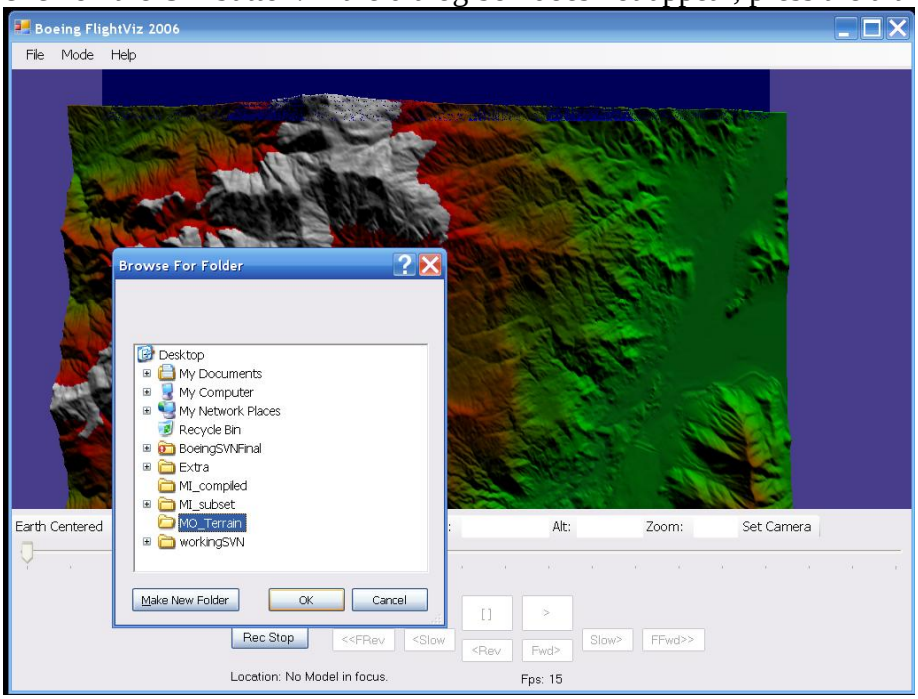
To add more terrain to the dataset, copy the GridFloat files into the "Terrain" directory. MapDataMaker must be used to split up large GridFloat elevation files, such as the one's provided by the USGS, so that FlightViz can use them. In addition a dataset folder can be loaded and used in real-time by using the Load Terrain menu option (see below).

### 3.2.4.2. Loading a dataset

To Load a terrain dataset click on File and then Load Terrain.

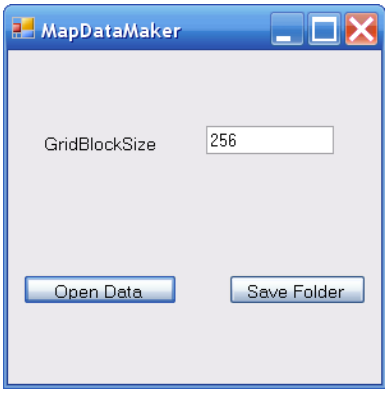



An open-folder dialog-box will pop-up, select the folder with the terrain dataset to be loaded and click on the OK button. If the dialog box does not appear, press the alt key to bring it into focus.



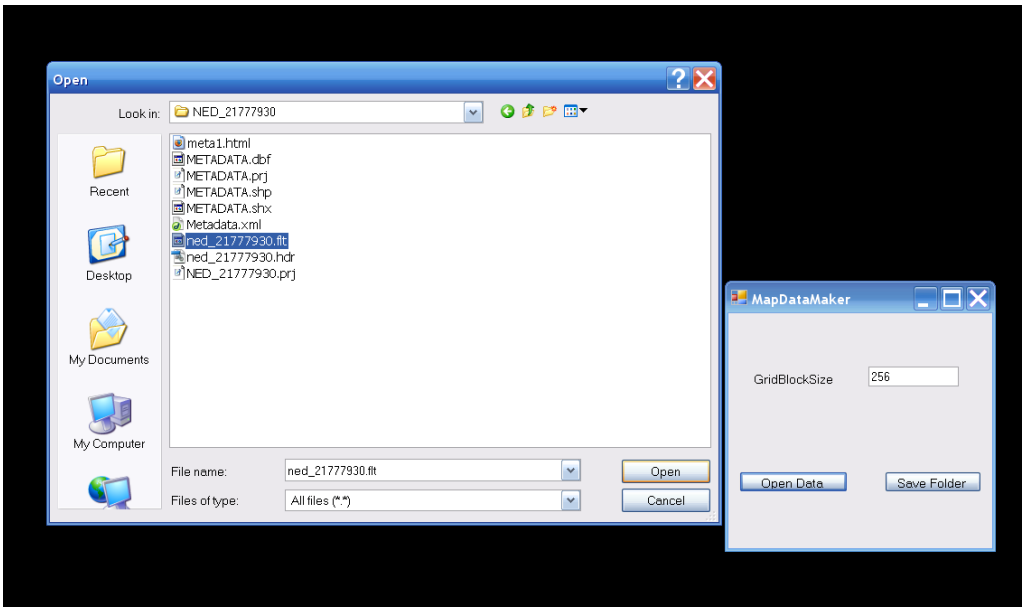
## 3.3. MapDataMaker

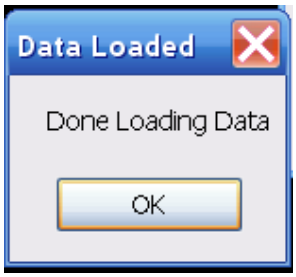
MapDataMaker is an application that loads elevation files in the GridFloat format and partitions the file into smaller blocks of a specified size.




### 3.3.1. Loading elevation data file

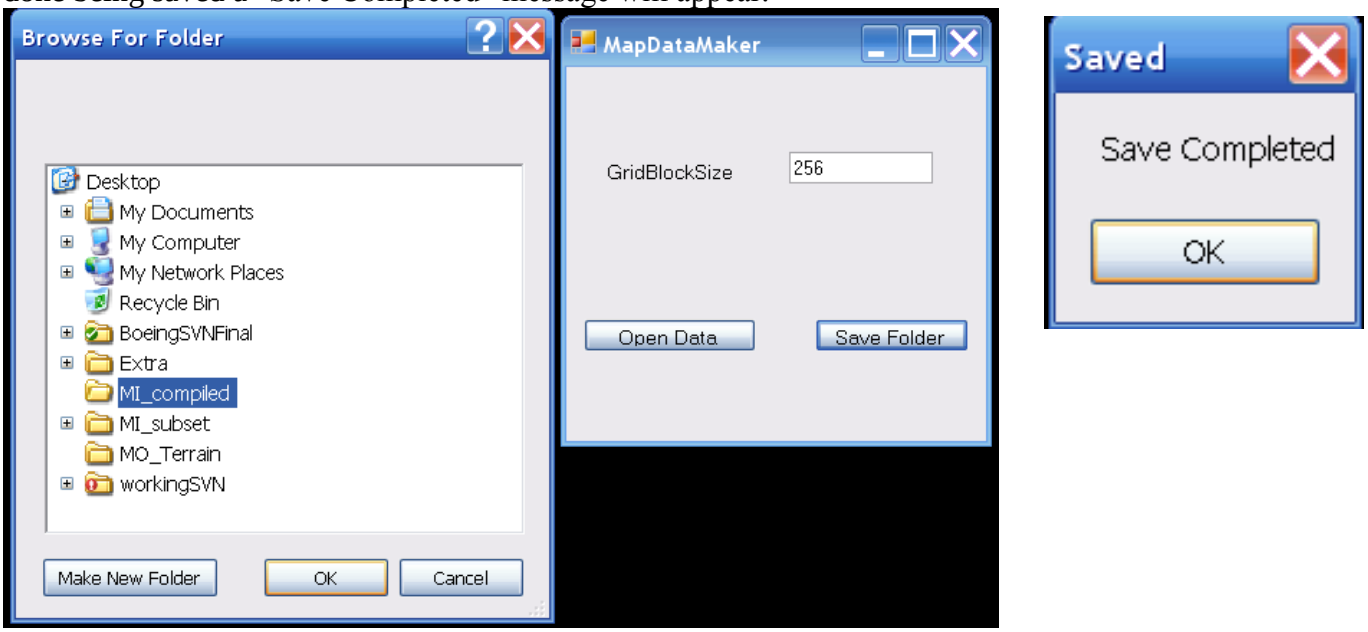
Click on the Open Data button to bring up an open-file dialog-box. Select the elevation data file to load and click on the Open button. A "Done Loading Data" message will pop-up once the elevation file is done loading.



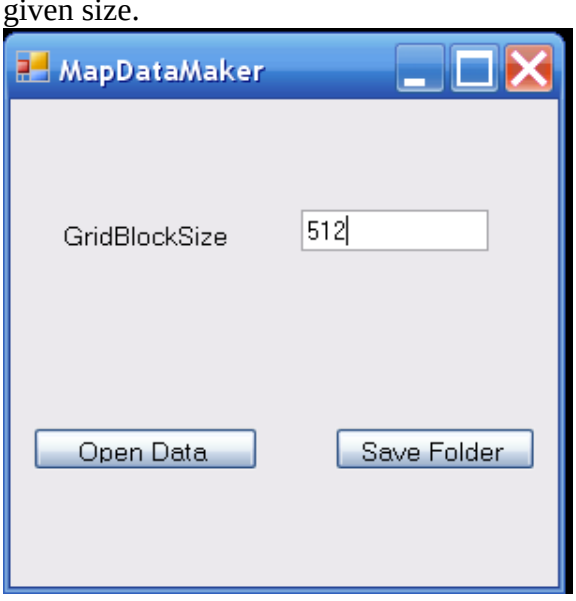
### 3.3.2. Saving Dataset to a folder

Click on the Save Folder button to save the partitioned blocks to a folder. Once the dataset is done being saved a “Save Completed” message will appear.




### 3.3.3. Changing the Grid Block Size

Click on the text field for GridBlockSize and type in the desired size for the partitioned blocks. Currently, FlightViz only supports the 256 GridBlockSize, but in the future, it can support any given size.



# 4. Administrator/Developer's Manual

## 4.1. TestServer

The DIS SimServer is a separate program from the FlightViz program used to test out the networking functionality of the FlightViz program. Currently it is hard coded to fly one or two planes in a circle at some location in California.

### 4.1.1. Networking

The TestServer currently does not use the DIS Framework to do the networking component of the server, as the DIS Framework's networking component does not contain any functionality for sending DIS PDUs over the wire. Instead the TestServer uses TCP/IP sockets provided in the System.Net.Sockets namespace. Currently it is set to connect to a FlightViz program that is running on the local machine with a loop-back address of 127.0.0.1 which is listening on port 4114. This is currently hard-coded. Changing this behavior requires modifying the source code.

#### 4.1.1.1. DIS PDU Serialization

The interface for serializing a PDU object has been generalized to write to a System.IO.Stream object. This allows flexibility in what the PDU gets written into; it can be a NetworkStream, a FileStream, or anything that derives from the Stream object. With this there are some performance considerations, especially when dealing with the NetworkStream. Passing a NetworkStream to the Serialize() function on a PDU results in a lot of small TCP packets being sent. While this does not violate the DIS protocol, it is rather inefficient, as most PDUs will fit in a single TCP/IP packet running on Ethernet. A better way to do it (as demonstrated in the TestServer program) is to serialize it to a MemoryStream, then send the MemoryStream to the NetworkStream, or directly to the appropriate socket, this typically results in a single TCP/IP packet being produced.

#### 4.1.1.2. DIS PDU Shutdown

Conditions for shutdown of the TCP/IP connection are one of the following:

- Any type of TCP/IP protocol error
- Any error in the DIS-PDU packet received
- End of transmission of packets
- FlightViz shutting down

The TestServer should gracefully handle any shutdown of the socket (usually wrapped in a try block).

## 4.2. DIS Framework

The DIS Framework consists of two components: the PDU component – which contains the PDU data structures and functions for serialization and deserialization of these data structures from Stream objects; and the Network component – which contains logic to listen for and deserialize PDU objects from the network using TCP/IP.

### 4.2.1. PDU

The PDU component contains data structures for the EntityState, Fire, and Detonate PDU types using the IEEE-1278-1995 standard. To gain assess to the DIS Framework, simply add a reference to the Boeing.FlightViz.Dis assembly in the project and utilize the Boeing.FlightViz.Dis.Pdu namespace.

#### 4.2.1.1. Adding new PDU types

Adding a new PDU type to the PDU component of the DIS Framework involves several steps. First, create a new class in the Dis.Pdu namespace following the naming conventions used in other PDU classes. The class must inherit from PduBase, which requires implementation of Serialize() and a pair of overloaded Deserialize() functions. Note that PduBase already includes the PduHeader object so there is no reason to add an additional PduHeader object. It is important to check that the PduHeader.PduType enumeration contains a correct value for the new PDU type. Any new fields within the new PDU should have classes created for them and should reside in the Dis.Pdu.Records namespace. Before creating new fields, ensure that it hasn't already been implemented, as there are fields that are common to many PDU types. Each PDU field type should be fully contained in one class and one file.

Once implemented the static PduFactory.Construct() method must be updated appropriately so that the Deserialization process can construct the new PDU type that is being added. Additionally, the code in the FlightViz, specifically the code in the SimEngine, must be modified to handle the new PDU type. This may also require changing some of the Model plug-in code, as the models are what determine how a PDU is to be used.

### 4.2.2. Networking Component

The networking component contains functionality to listen on a specific TCP/IP port, accept connections, and deserialize PDUs taken off the network. There are two versions of the functionality: one is synchronous, and one is asynchronous, however both are accessed using the same interface:

Start() – starts the client listening for new packets
CheckForNewPdus() – checks to see if there are any new PDUs awaiting since Start() or CheckForNewPdus() was called
Stop() – shuts down the listener

The Asynchronous and Synchronous versions each have different performance characteristics. The Synchronous does better on uniprocessor systems, as there is less overhead for switching contexts. It scales with processor speed, however it does not scale with additional processors. The Asynchronous version does better on multi-processor systems, and scales with both speed of the processors and number of cores. To scale with the number of cores requires that the OS have a multi-threaded TCP/IP stack, which is not available until Windows Vista. Windows XP and before will scale with processors, but will have a limit with the TCP/IP stack being serialized. Currently the Synchronous version is hard-coded and must be changed by modifying the SimEngine constructor to create an AsyncSimListener instead of a SynchronousSimListener.

## 4.3. FlightViz

### 4.3.1. Models

FlightViz models are .NET assemblies that are loaded at runtime from the "Models" directory. They must conform to the SimEngine plugin-format.

#### 4.3.1.1. Creating a Model Plug-In

Creating a Model Plug-in involves creating a new .NET class library that implements the IModelPlugin interface for the whole assembly, and the IModel interface for each model that the assembly is capable of handling. The IModelPlugin interface is how FlightViz accesses the functionality of a Model Plug-In and may implement one or more models. Interface notes:

- InitializePlugin() –will be called once when the plugin is first loaded, a Valid Direct3D device will be passed. This is meant to be used as a place where static data structures can be initialized and models can be loaded.
- PluginSupports() – will be called each time a new PDU is seen by FlightViz, plugin should examine the passed in EntityStatePdu packet and determine if a model contained within the plugin can consume this type of entity, a model can be partially supported.
- InstantiateModel() - will instantiate a new IModel object that reflects the type of the EntityStatePdu passed in. If PluginSupports() returns not supported, and the EntityStatePdu packet sent in is not supported, the result of this function should be to return a null value.

The IModel interface should include the following:

- AddPdu – should quickly store the latest PDU. Internal updates to the state of the PDU should be updated in the Update() function, this may be called multiple times with multiple PDUs before an Update() call occurs
- Update – should update the internal state of the model, this is called before the Render() function and after the AddPdu() function
- Render – this function should render the model
- Id – this is a property that should reflect the Id of the model as given by the EntityStatePdu Packets.

#### 4.3.1.2. 3D Model Format

The model should be at the location (0, 0, 0) in the 3D modeling program and oriented to a right handed coordinate system. The model should be facing in the positive Y direction with Z pointing up. The units for the model should be in meters. The Model Plug-In allows a model to be in any file format, as long as the appropriate render and IntializePlugin () functions are created that load that format. The F-18 A/B and Mig29 Model Plug-Ins developed for FlightViz both use the .X model format.

## 4.3.2. Terrain

In FlightViz the handling of terrain is split into two classes, Terrain and TerrainBuffer. The Terrain class deals with how to load an individual terrain block into memory and render it. The TerrainBuffer class manages what terrain blocks to load and unload.

### 4.3.2.1. Supporting more data formats

FlightViz currently supports the GridFloat format but can be extended to support more formats such as BIL, GeoTIFF, and ArcGrid. In order to do this the main information needed from the elevation data format is the number of rows and columns, cell size in degrees, and the actual elevation values. BIL format would be the easiest to add, as it is very similar to GridFloat except the data is given as integers instead of floating-point values. The only class that would need changing is the Terrain file and the only function that would need changing would be the constructor.

### 4.3.2.2. Changing terrain behavior variables

#### *4.3.2.2.1. Changing the render view size*

To change the render view size change the variable myGridSize in the TerrainBuffer constructor to the desired amount. A size of three was chosen as the default amount to render as it worked well with the machines that were tested on. A larger value will require more memory and system performance. Odd integer values work best.

#### *4.3.2.2.2. Changing the cell size*

To change the cell size used by FlightViz go to the TerrainBuffer constructor and change the variable myCellSize to the amount needed. The cell size used in FlightViz should match the cell size that is in the source elevation data that is to be used.

#### *4.3.2.2.3. Changing the grid block size*

The grid block size is the number of rows and columns that each file will have. Only one value is needed because all files must be square. To change the grid block size used by FlightViz go to the TerrainBuffer constructor and change myBlockSize to the desired amount. The grid block size used in FlightViz should match the grid block size that the elevation to be used has.

#### *4.3.2.2.4. Changing the color scale*

The color gradient for the terrain is set in the Buffer() function in the Terrain class. The current gradient works by setting the lowest elevations to be blue and the highest elevations to be white. The maximum height for blue can be set by changing maxBlue and the minimum height for white can be set by changing minWhite to the value desired.

#### *4.3.2.2.5. Changing the max buffer distance*

To change the max amount of terrain blocks that FlightViz buffers away from the center location change the variable maxBufferDistance in the TerrainBuffer constructor to the amount desired. Any blocks outside of the max buffer distance will be removed.

## 4.4. MapDataMaker

MapDataMaker uses the TerrainSplitter class to load up data, partition up the data and save the partitioned data set. Within the TerrainSplitter class, LoadData() loads up the source elevation file. The CreateBlocks() function breaks up the data into separate equally sized, square blocks. The SaveData() function saves each created block to a separate file to a specified folder.

### 4.4.1. Extending supported data formats

MapDataMaker can be extended to support additional formats in two ways. By extending the application to load more file formats and by extending it to save to more formats. To add support for loading additional formats, LoadData() in TerrainSplitter would have to be modified to get the number of columns and rows, cells size, and a NODATAValue would have to be decide upon. This would allow for the program to load additional formats while still being able to save to GridFloat files for FlightViz. To expand MapDataMaker to be able to save to more formats, the SaveData() function in TerrainSplitter would have to be changed to be able to save in another format. SaveData() would have to be changed to write to the specification of the desired format.