Gulfstream Aerospace Aircraft Static Position Utility

Mark Eugene Ray

Follow this and additional works at: https://digitalcommons.georgiasouthern.edu/etd_legacy

Part of the Aerospace Engineering Commons, and the Aviation Commons

Recommended Citation
https://digitalcommons.georgiasouthern.edu/etd_legacy/258

This thesis (open access) is brought to you for free and open access by Digital Commons@Georgia Southern. It has been accepted for inclusion in Legacy ETDs by an authorized administrator of Digital Commons@Georgia Southern. For more information, please contact digitalcommons@georgiasouthern.edu.
GULFSTREAM AEROSPACE
AIRCRAFT STATIC POSITION UTILITY

Mark Eugene Ray
GULFSTREAM AEROSPACE
AIRCRAFT STATIC POSITION UTILITY

A Thesis
Presented to
The College of Graduate Studies of
Georgia Southern University

In Partial Fulfillment
of the Requirements for the Degree
Master’s of Science
In the Department of Mathematics and Computer Science

by
Mark Eugene Ray
May 2001
April 12, 2001

To the Graduate School:

This thesis entitled, "Gulfstream Aerospace Aircraft Static Position Utility", and written by Mark Eugene Ray is presented to the College of Graduate Studies of Georgia Southern University. I recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science in the Department of Mathematics and Computer Science.

Dr. Chong-Wei Xu, Supervising Committee Chair

We have reviewed this thesis and recommend its acceptance:

Dr. Goran Lesaja, Committee Member  

Dr. Ahmed Barbour, Committee Member  

Dr. Don Fausett, Department Chair

Accepted for the College of Graduate Studies

G. Lané Van Tassell  
Dean, College of Graduate Studies
DEDICATION

I dedicate this thesis to my wife Julie and daughter Maggie for their patience, support and understanding during the three years it has taken me to finish the requirements for this degree.
ACKNOWLEDGMENTS

I would like to thank Dr. Chong-Wei Xu and Mr. David MH Goyette for the assistance and insight they provided for methods implemented within the GUI and communication components of the Gulfstream Aircraft Static Position Utility. My thanks also go to Mr. Doug McKissack for his assistance with certain computational components. Finally, I would like to thank Mrs. Tanya DeVoe for her assistance and guidance with the text processing applications that were utilized to prepare and support this project. All of these individuals made valuable contributions that enhanced the efficiency, functionality and presentation of this project.
ABSTRACT

GULFSTREAM AEROSPACE AIRCRAFT STATIC POSITION UTILITY

May 2001

Mark Eugene Ray

M.S. Georgia Southern University
B.S. Georgia Southern University

Directed by: Professor Chong-Wei Xu

My research project implements a Java based utility developed for commercial application by Gulfstream Aerospace employees and customers to determine static position data of Gulfstream aircraft. The static position data includes aircraft attitude, nose and main landing gear loads and aircraft ground clearance data.

The previous methods used to obtain this data were a family of spreadsheets. The spreadsheet solution has proven to be awkward and inaccessible for providing the required data. Therefore, Gulfstream Aerospace Management approved the design and implementation of a new aircraft static position utility.

The new utility is identified as the Gulfstream Aircraft Static Position (GASP) Utility. The GASP Utility dynamically provides a user with the static aircraft position information as a function of aircraft mass and landing gear parameters. The client side of the utility is an Internet accessible Java Swing applet that is used to request, edit, display and plot aircraft static position data. The applet is combined with a Java application file server that reads and returns aircraft data as requested by the client. The aircraft data is located in text files on the server computer. The applet
communicates with the server using Remote Method Invocation (RMI) to obtain the requested aircraft data file. The user is able to interactively adjust the aircraft configuration using graphical or text input and dynamically view the resulting aircraft static position, gear loads and ground clearance data.

This project report will present several aspects of the GASP Utility including the Business Case, the technologies used, the detailed development of the file server, client and Graphical User Interface (GUI), the communication and computational methods applied and finally a users guide.
# TABLE OF CONTENTS

**DEDICATION** .......................................................... iii

**ACKNOWLEDGMENTS** ................................................. iv

**ABSTRACT** ........................................................... v

**LIST OF TABLES** ...................................................... ix

**LIST OF FIGURES** .................................................... x

**CHAPTER**

I. **INTRODUCTION** .................................................... 1
   A. Business Case ...................................................... 2
   B. The Gulfstream Aircraft Static Position Utility .............. 3
   C. The Graphical User Interface Design ........................... 5
   D. Technologies Used by the Utility .............................. 7
      1. Internet and the World Wide Web ......................... 8
      2. Java ............................................................. 8
      3. Applets ........................................................ 9
      4. Swing .......................................................... 10
      5. Reflection .................................................... 10
      6. TCP/IP .......................................................... 11
      7. RMI ............................................................. 12
   E. Project Outline .................................................. 14

II. **DETAILED GASP UTILITY DESIGN** .............................. 15
   A. Architecture ..................................................... 16
   B. Server and Client Class Selection ............................ 17
   C. Server Design ................................................... 18
   D. Client Design ................................................... 19
      1. Client Classes ............................................... 20
      2. Client GUI .................................................... 22
      3. GUI Panel/Tab Descriptions ................................. 23

III. **COMMUNICATIONS** ............................................... 31
    A. TCP/IP ............................................................ 32
       1. Layer 1 - Network Access Layer .......................... 32
       2. Layer 2 - Internet Layer ................................. 32
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3. Layer 3 - Transport Layer</td>
<td>34</td>
</tr>
<tr>
<td>4. Layer 4 - Application Layer</td>
<td>35</td>
</tr>
<tr>
<td>B. RMI</td>
<td>36</td>
</tr>
<tr>
<td>1. Server Interface and Implementation</td>
<td>39</td>
</tr>
<tr>
<td>2. Client RMI Implementation</td>
<td>41</td>
</tr>
<tr>
<td>3. Compile and Execute</td>
<td>41</td>
</tr>
<tr>
<td>IV. COMPUTATIONAL METHODS</td>
<td>42</td>
</tr>
<tr>
<td>A. Landing Gear Description</td>
<td>43</td>
</tr>
<tr>
<td>B. Aircraft Balance</td>
<td>44</td>
</tr>
<tr>
<td>C. Solution Methods</td>
<td>45</td>
</tr>
<tr>
<td>V. USER'S GUIDE</td>
<td>51</td>
</tr>
<tr>
<td>A. Starting the Server</td>
<td>51</td>
</tr>
<tr>
<td>B. Baseline Aircraft Data File Modification</td>
<td>52</td>
</tr>
<tr>
<td>C. Starting the GASP Utility (Client Applet)</td>
<td>52</td>
</tr>
<tr>
<td>D. Selecting and Loading Aircraft Data</td>
<td>53</td>
</tr>
<tr>
<td>E. Modifying Data from within the Utility</td>
<td>54</td>
</tr>
<tr>
<td>F. Utility Termination</td>
<td>54</td>
</tr>
<tr>
<td>VI. SUMMARY</td>
<td>55</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>56</td>
</tr>
<tr>
<td>APPENDICES</td>
<td>57</td>
</tr>
<tr>
<td>APPENDIX A: SAMPLE INPUT DATA</td>
<td>58</td>
</tr>
<tr>
<td>APPENDIX B: SOURCE CODE VARIABLE DESCRIPTIONS</td>
<td>65</td>
</tr>
<tr>
<td>APPENDIX C: SOURCE CODE</td>
<td>70</td>
</tr>
<tr>
<td>1. ACDataServer.java</td>
<td>70</td>
</tr>
<tr>
<td>2. ACDataServerImpl.java</td>
<td>71</td>
</tr>
<tr>
<td>3. StaticGear.java</td>
<td>73</td>
</tr>
<tr>
<td>4. ACdata.java</td>
<td>88</td>
</tr>
<tr>
<td>5. Gear.java</td>
<td>103</td>
</tr>
<tr>
<td>6. StrokeDist.java</td>
<td>113</td>
</tr>
<tr>
<td>7. WtCGcanvas.java</td>
<td>121</td>
</tr>
<tr>
<td>BIBLIOGRAPHY</td>
<td>125</td>
</tr>
</tbody>
</table>
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>I. Previous Aircraft Static Position Utility Issues</td>
<td>2</td>
</tr>
<tr>
<td>II. New Aircraft Static Position Utility Objectives</td>
<td>3</td>
</tr>
<tr>
<td>III. Project Outline of Remaining Chapters</td>
<td>14</td>
</tr>
<tr>
<td>IV. Example Application Layer Services</td>
<td>35</td>
</tr>
<tr>
<td>V. URL Components</td>
<td>36</td>
</tr>
<tr>
<td>VI. GVdata.txt</td>
<td>58</td>
</tr>
<tr>
<td>VII. Source Code Variable Descriptions</td>
<td>65</td>
</tr>
<tr>
<td>VIII. Source Code Listing</td>
<td>70</td>
</tr>
</tbody>
</table>
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Number</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Static Position Information</td>
<td>1</td>
</tr>
<tr>
<td>2.</td>
<td>Basic GASP Utility Architecture</td>
<td>4</td>
</tr>
<tr>
<td>3.</td>
<td>GUI Design</td>
<td>5</td>
</tr>
<tr>
<td>4.</td>
<td>Initial Appearance of Utility</td>
<td>6</td>
</tr>
<tr>
<td>5.</td>
<td>Java Applets</td>
<td>10</td>
</tr>
<tr>
<td>6.</td>
<td>Client Server Connection</td>
<td>12</td>
</tr>
<tr>
<td>7.</td>
<td>RMI System</td>
<td>13</td>
</tr>
<tr>
<td>8.</td>
<td>GASP Utility File System</td>
<td>15</td>
</tr>
<tr>
<td>9.</td>
<td>Utility Class Description (UML)</td>
<td>17</td>
</tr>
<tr>
<td>10.</td>
<td>Client Design - GUI Layout</td>
<td>18</td>
</tr>
<tr>
<td>11.</td>
<td>Client Class Description (UML)</td>
<td>19</td>
</tr>
<tr>
<td>12.</td>
<td>Initial Appearance of Utility</td>
<td>22</td>
</tr>
<tr>
<td>14.</td>
<td>AC Data Tab</td>
<td>24</td>
</tr>
<tr>
<td>15.</td>
<td>Parameters Tab</td>
<td>25</td>
</tr>
<tr>
<td>16.</td>
<td>Results Tab</td>
<td>26</td>
</tr>
<tr>
<td>17.</td>
<td>Geometry Tab</td>
<td>27</td>
</tr>
<tr>
<td>18.</td>
<td>Service Tab</td>
<td>28</td>
</tr>
<tr>
<td>19.</td>
<td>Gear Stroke Tab</td>
<td>29</td>
</tr>
<tr>
<td>20.</td>
<td>Distance to Ground Tab</td>
<td>30</td>
</tr>
<tr>
<td>21.</td>
<td>Communication in the GASP Utility</td>
<td>31</td>
</tr>
</tbody>
</table>
List of Figures (Continued)

22. TCP/IP Layers .......................................................... 32
23. Marshaling Example ..................................................... 37
24. RMI Registry ............................................................. 37
25. RMI System .............................................................. 39
26. Landing Gear Location .................................................. 42
27. Main Gear Major Components ........................................ 43
28. Example Aircraft Loading .............................................. 44
29. Coordinate Systems and Deck Angle .............................. 47
30. Ground Clearance ...................................................... 50
31. Initial Appearance of Utility ........................................... 53
Chapter I

INTRODUCTION

Gulfstream Aerospace is a manufacturer of long-range business class aircraft. The company has approximately 8,000 employees and supports over one thousand aircraft of several different models ranging from the G-I through the G-V. The aircraft operators and company employees frequently have questions about the static ground position of these aircraft. The static position data includes aircraft attitude, nose and main landing gear loads and aircraft ground clearance data as illustrated in Figure 1 below. This data is primarily a function of the aircraft weight, center of gravity and landing gear parameters. The numerical analysis required to determine the static position data involves over 100 parameters and a nonlinear convergence problem. The results of a static position analysis must be accessible in a timely manner to the appropriate people.

**Figure 1: Static Position Information**
Business Case

Until January 2001, the only way to access the static position data was by using a series of undocumented Microsoft Excel spreadsheets (one spreadsheet for each aircraft model). The spreadsheets were maintained and used primarily by a single engineer. Each spreadsheet contains the base data for a given aircraft model and includes the required numerical analysis equations to provide basic aircraft static position results. Since its implementation, the spreadsheet solution has proven to be awkward and inaccessible for providing the required data. Some of the issues discovered while using the spreadsheet solution are listed in Table I.

Table I: Previous Aircraft Static Position Utility Issues

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The spreadsheets can be copied or distributed with little control over the final destination and integrity of the analysis or the data.</td>
</tr>
<tr>
<td>2</td>
<td>The spreadsheets have limited access. Request for an aircraft static position analysis must be made by phone or e-mail to the engineer that maintains the spreadsheets.</td>
</tr>
<tr>
<td>3</td>
<td>A different copy of the spreadsheet is used for each aircraft type. Therefore, if the analysis needs an update, several spreadsheets would need to be modified.</td>
</tr>
<tr>
<td>4</td>
<td>Input of aircraft data and the presentation of results are static. The spreadsheet cells are input by hand.</td>
</tr>
</tbody>
</table>

Based on the Table I issues for the spreadsheet analysis, Gulfstream Aerospace Management has approved the design and implementation of a new aircraft static position utility. Meetings with the management and the appropriate engineering staff were held to derive a list of objectives for the new utility. A summary of these objectives are provided in Table II.

With the objectives for the utility established and management support within the Gulfstream engineering organization, a new aircraft static position utility was designed and implemented.
Table II: New Aircraft Static Position Utility Objectives

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Internet Access: Anyone with the required access information can use the utility from a web browser.</td>
</tr>
<tr>
<td>2</td>
<td>Maintenance: The utility’s source code and data must remain under the control of the Gulfstream Loads and Dynamics Group.</td>
</tr>
<tr>
<td>3</td>
<td>Security: All source code and data files must be maintained in a secure directory on the server computer.</td>
</tr>
<tr>
<td>4</td>
<td>GUI: Presentation of the input parameters and resulting solutions must be complete, logical and dynamic (real time updates).</td>
</tr>
<tr>
<td>5</td>
<td>Flexibility: Aircraft data presented by the utility can be easily updated for trend studies.</td>
</tr>
<tr>
<td>6</td>
<td>Efficient: It must be easy for a user to load other aircraft models or reset the current model to the published initial values.</td>
</tr>
<tr>
<td>7</td>
<td>Transferable Technology: The methods used to resolve the aircraft static position analysis should be transferable to similar projects that need improved implementations.</td>
</tr>
</tbody>
</table>

The Gulfstream Aircraft Static Position Utility

The new utility is identified as the Gulfstream Aircraft Static Position (GASP) Utility. The utility must dynamically provide a user with the static position information as a function of aircraft mass and landing gear parameters. A Java Applet was selected as the platform for the GASP Utility to accomplish this goal. Java is a platform independent computer programming language developed by Sun Microsystems. An Applet is a special type of Java program intended for implementations on the Internet. The Java Applet platform was selected because of the accessibility from a web browser over the Internet and the ability to present dynamic content. The basic architecture of the utility is presented in Figure 2.

As seen in the figure, all components of the utility including the Applet, file server and aircraft data are located on the host computer in a secure directory within the file system of the web server (Pt. 1). In this location, only system administrators with the proper access can modify the utility components. Normal
access to the utility is through a web browser. Therefore, when a user needs utility access, a web browser must be opened on a client computer (Pt. 2). When the proper web page is located and the Hyper-Text Markup Language (HTML) link to the Applet is selected (Pt. 3), a copy of the Applet utility is transmitted to the client using Transmission Control Protocol/Internet Protocol (TCP/IP) (Pt. 5). The TCP/IP is covered in detail in Chapter III. The Applet is then loaded into the browser and presented in a new window (Pt. 4). After the Applet has started, the user must select an aircraft model to initialize the utility. The initial aircraft data is transmitted from the server to the client (Pt. 5) using Remote Method Invocation (RMI) which is covered in detail in Chapter III.

**Figure 2: Basic GASP Utility Architecture**

The architecture alone satisfies several of the utility's objectives. Unlimited, simultaneous access to the utility is available from a web browser. The utility's source code and baseline aircraft data remain secure on the server. The use of an Applet allows the user interface to be interactive and dynamic.
The Graphical User Interface Design

The Graphical User Interface (GUI) design of the utility Applet must be user friendly and logically arranged. The GUI is assembled using Java Swing components. Swing components are new Java elements that provide enhanced functionality and easier implementation of GUI designs. The utility GUI is divided into six major panels using the configuration of an aircraft as seen in Figure 3.

Figure 3: GUI Design

The major panels include one for the primary aircraft data (Panel 1 on Figure 3), one for the graphical presentation of aircraft mass properties (Panel 5) and one for ground clearance information (Panel 3). The remaining three sections present detailed landing gear data for the port (Panel 4), nose (Panel 2) and starboard (Panel 6) landing gear. Note that the terms port and starboard are
aviation terms equivalent to left and right, respectively, if one was looking out the front of the aircraft. Detailed descriptions of each panel are provided in Chapter II.

Since the utility is a Java Applet, it may be initiated from any web browser with access to the HTML file link. Once the Applet is loaded, the user must select an aircraft model to work with using a drop down selection tool in Panel 1. The identity of the selected aircraft is sent to the file server that reads a text file containing the required information about the selected aircraft. The data is returned to the Applet (client) where all variables and graphical data are initiated. The initial appearance of the Applet window once an aircraft model is loaded is presented as Figure 4.

**Figure 4: Initial Appearance of Utility**

At this point, the user may graphically select the aircraft's zero fuel gross weight and center of gravity by dragging or clicking the mouse anywhere on the
mass properties canvas (Pt. 6 on Figure 4). Fuel is added to the aircraft using the vertical slider (Pt. 7) located in the aircraft data panel. A user may also update any aircraft information by modifying the editable text fields. Any white text field may be modified at any time (Pt. 8). The gray text fields are for presenting results and are not user editable (Pt. 9). All results are updated dynamically as the mass properties or other parameters are modified. The user may reset the current data to the initial conditions at any time by re-selecting the current aircraft model using the drop down selection tool (Pt. 10) in the aircraft data panel. In addition, a new aircraft model can be selected at any time.

**Technologies Used by the Utility**

The GASP Utility requires the use of many technologies in order to obtain the project objectives. The utility is accessible to users from any web browser using the World Wide Web and its' supporting infrastructure and protocols. The client and server components of the utility were implemented using only Java technologies. The client is a Java Swing Applet that is accessed by an HTML link on the Gulfstream Engineering home page. Once the client Applet is loaded, it needs to contact the server to access text files that define the initial aircraft data. The server is a Java application that is executed on the host computer and remains running at all times. The text files needed by the client contain detailed data about a specific aircraft type. A string passed from the client Applet to the server identifies the appropriate text file. The file contents are loaded into a Java HashMap object that is returned to the client for use. A HashMap is a Java data structure that stores data as a list of key and value pairs. The client uses RMI and reflection technologies to communicate with the server and initialize or update the client state using the contents of the HashMap. A brief description of the primary technologies used by the GASP Utility is provided below.
Internet and the World Wide Web

The Internet provides the infrastructure over which the utility communicates. The Internet is a multi-node communication grid created by the U.S. Military in the 1960s. The idea was to provide multiple paths along which data could travel. That way, if connections between nodes were broken, the data could find several other paths to its destination. As the communication needs of the military started to rely on other technologies, the Internet infrastructure was essentially turned over to the public. In recent years, the Internet has grown at a phenomenal rate connecting hundreds of millions of computer systems worldwide. The most popular implementation on the Internet is the World Wide Web (WWW) that allows fast, easy multimedia content on virtually any subject. Today, WWW sites are maintained for most companies, universities, organizations and even millions of individuals [2].

Java

Java is the name given to an open source, hardware independent and operating system independent programming language developed by Sun Microsystems. The Java syntax is based on the best aspects of C and C++ that are two of the most common programming languages used today. This fact makes Java usable to the current developers of most types of software implementations from operating systems to personal computer applications. Java was designed to be truly portable so it is ideal for Internet and World Wide Web based applications such as the GASP Utility. Java has "built in the features people really need such as strings, graphics, graphical user interface components, exception handling, multithreading, multimedia (audio, images, animation and video), file processing, database processing, Internet and World Wide Web-based client/server networking and distributed computing, and prepackaged data structures [2]." The
ability to give the WWW dynamic content is one of the main aspects of Java that was originally promoted. However, as Java matures, it has the potential to become one of the most popular general purpose programming languages in the world [2].

Java applications have the ability "to use the Internet and world wide web to interact with other applications, file systems and databases. These capabilities allow Java programmers to develop the commercial, distributed applications used in industry today [2]."

Applets

An Applet is a special type of Java program intended for implementations using the Internet as seen in Figure 5. Applets run in World Wide Web browsers like Netscape Communicator or Microsoft Internet Explorer (Pt. 11 on Figure 5) by clicking on the HTML file link associated with the Applet on the client computer. Applets provide the ability to add powerful dynamic content to WWW sites by using real time graphics and GUI components. Applets have almost identical functionality to Java applications with one important exception. Since Applets can be loaded and executed over a network, they have security provisions not found in Java applications. The two main restrictions are that an Applet can't normally access files (read or write) on the computer that it's executing on (Pt. 12), and an Applet cannot make network connections except to the host from which it originated (Pt. 13) [2]. Therefore, the GASP Utility can only access the aircraft data files if it is loaded from the host that maintains the utility and data files (Pt. 14). An Applet can also be located, loaded and executed on the same computer. In this case, the computer (Pt. 15) will be the client and the server (local host). However, the Applet will not be able to connect to any network outside of the local machine (Pt. 16).

Dynamic content and security are two properties of the Java applet technology that helped satisfy many of the utility design objectives presented in Table II.
Figure 5: Java Applets

Swing

Swing components are one of a group of features included in the Java Foundation Classes (JFC). The Swing package includes a large selection of pre-defined GUI components such as buttons, text fields, scroll panes, tabbed panes and slider bars just to name a few. These GUI components provide for enhanced functionality and easier implementation of Java GUI designs [2]. The GUI implementation of the GASP Utility uses many swing components.

Reflection

Reflection, in the context of Java, refers to the reflection Application Programming Interface (API). This API is very powerful because it provides the ability to represent, or reflect, the classes, interfaces, and objects in the current
Java Virtual Machine (JVM) [3]. The JVM interprets Java technology bytecodes so that the operating system on this computer can understand them. In the GASP Utility, reflection is used to very efficiently initialize all text fields, text field listeners and most variables each time an aircraft data file is loaded. Reflection is also used in the utility to update variables any time a text field value is modified. Details on how the reflection API is used in this utility are provided in Chapter II.

TCP/IP

The Transmission Control Protocol/Internet Protocol (TCP/IP) is the most popular family of protocols in use today. A protocol in the context of computers and networking is just an agreed upon set of rules for communicating. TCP/IP is not vendor specific and has been implemented on almost everything including Local Area Networks (LANs) and the WWW. The family of TCP/IP protocols can be divided into four major layers. The layers are the Application, Transport, Internet and the Network Access Layers. The Internet Layer contains the Internet Protocol (IP), which is one of the more important protocols. It is responsible for controlling the flow of data through a network by providing a standardized format for data and selecting the best data transmission route. Both the GASP Utility and the aircraft data it uses are transmitted to the client computer using this protocol among others. The basic connection architecture is presented in Figure 6. The IP address and port number will identify a unique connection within a computer [1]. Therefore, if the client (Pt. 17 on Figure 6) knows the IP address (Pt. 18) and the appropriate port number (Pt. 19) of the Server computer (Pt. 20), then a network connection over the Internet (Pt. 21) can be established. A detailed description of the TCP/IP protocols as it relates to the GASP Utility is provided in Chapter III.
Remote Method Invocation (RMI) is used to provide communication between the Applet client and server application as presented in Figure 7. "RMI allows Java objects running on the same or separate computers to communicate with one another via remote method calls. Such method calls appear the same as those operating on objects in the same program [2]." Using this technology, a method located in the server application (Pt. 22 on Figure 7) can be called by the client applet (Pt. 23) to load and return the aircraft data (Pt. 24) as if the method were in the client program. To accomplish this, the RMI system requires a registry service (Pt. 25) be running on the server computer. The registry maintains a database of the available remote objects (method calls). The client must obtain a reference to the remote object from the registry before that object can be used.
(Pt. 26 and 27 on Figure 7). The RMI system serializes the Java string (Pt. 28) or HashMap object (Pt. 29) for transmitting over the network (Internet) and then de-serializes them for use in the destination applet or application [2]. A detailed description of the RMI system as it relates to the GASP Utility is provided in Chapter III.

**Figure 7: RMI System**
Project Outline

A brief description of the remaining chapters in this project is presented in Table III below.

Table III: Project Outline of Remaining Chapters

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title / Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Detailed Utility Design / provides a detailed description of the architecture, design and graphical user interface (GUI) of the utility</td>
</tr>
<tr>
<td>3</td>
<td>Computational Methods / describes the analytical methods used to calculate the aircraft static position data including the deck angle, landing gear loads and ground clearance data</td>
</tr>
<tr>
<td>4</td>
<td>Communication Methods / provides details about the communication technologies TCP/IP and RMI which are used by the GASP Utility</td>
</tr>
<tr>
<td>5</td>
<td>User's Guide / describes how to access and operate the GASP Utility</td>
</tr>
<tr>
<td>6</td>
<td>Summary / summarizes the design, implementation and deployment of the GASP Utility</td>
</tr>
</tbody>
</table>
Chapter II

DETAILED GASP UTILITY DESIGN

The GASP Utility was designed to meet the objectives provided in Table I while being user friendly, logically arranged and technically correct. The major aspects of the utility that contributed to these goals were the architecture, server design, client design, and the GUI.

Figure 8: GASP Utility File System
Architecture

The basic architecture of the GASP Utility was reviewed in Chapter I using Figure 2. A more detailed description of the architectural elements is provided in this chapter. First, Figure 8 above presents the file system structure of the utility on the server computer. All files which make up the utility are located in a single directory (Pt. 29 on Figure 8) within the file system of the web server (Pt. 30). These files include the source (*.java), executable (*.class), aircraft data and the applet HTML files. An engineering web page (Pt. 31) currently contains the link to the HTML file (Pt. 32) that executes the utility applet (Pt. 33).

Before the applet can be used, the RMI registry and aircraft data file server for the utility must be activated. The file server is responsible for reading and transmitting aircraft data files when requested by the client(s). The server is a Java application that uses Remote Method Invocation (RMI) to communicate with the client(s). The RMI technology was introduced in Chapter I and will be presented with more detail in Chapter III. The RMI registry and data server are intended to remain running at all times listening for data requests from client applets for aircraft data.

Once the applet link is selected, the applet class files are transmitted to the requesting client computer and loaded into the browser. The applet is transmitted from the server to the client over the Internet using the TCP/IP which is discussed in detail in Chapter III. The browser has access to the local Java Runtime Environment (JRE) which compiles the applet class files into an executable file compatible with hardware and operating system of the local machine. The applet is then executed in the web browser and communication is established back to the server using the RMI technology.
Server and Client Class Selection

A detailed listing of the classes that define the GASP Utility and their associated methods is provided in Figure 9 using the Universal Markup Language (UML) notation. The server implementation class "ACDataServerImpl" (Pt. 34 on Figure 9) is a Java application which extends the "UnicastRemoteObject" required for using RMI and implements the interface "ACDataServer" (Pt. 35). The interface only defines one remote method "getAircraftInfo" which is defined in the server implementation and called from the client applet (Pt. 36). The client applet is implemented using a JApplet class and has four support classes (Pt. 37) that are "ACData", "Gear", "WtCGcanvas", and "StrokeDistance". Details on the client classes is provided later in this chapter.

![Figure 9: Utility Class Description (UML)]
Server Design

The only responsibility of the GASP Utility file server is to register a remote object with the RMI registry then read and return aircraft data files as requested by the client(s). The server is a Java application with one constructor and three methods (P1. 34 on Figure 9). The main method just creates an object "aircraft" of type "ACDataServerImpl" and binds it to the RMI registry. The method "getAircraftInfo" of type HashMap is the remote method on the server called by the client. It calls the server method "updateAircraftData" also of type HashMap that reads the requested aircraft data file, loads the data into a HashMap and returns the HashMap to the client. The complete listing of the server source code files "ACDataServer.java" and "ACDataServerImpl.java" are provided in Appendix C.

Figure 10: Client Design - GUI Layout
Client Design

The client is a Java applet built around a GUI divided into six major panels using the configuration of an actual aircraft as seen in Figure 10. The major panels were defined in Chapter I and are repeated here for convenience. The panels include one for the primary aircraft data (Panel 1 on Figure 10), one for the graphical presentation of aircraft mass properties (Panel 5) and one for ground clearance information (Panel 3). The remaining three sections present detailed landing gear data for the port (Panel 4), nose (Panel 2) and starboard (Panel 6) landing gear.

Figure 11: Client Class Description (UML)
Client Classes

The relationship between the classes presented in Figure 9 and the GUI layout in Figure 10 is shown in Figure 11 above. The letter given in each panel represents the instantiation of the indicated GUI support class. The four GUI classes (Pt. 38 on Figure 11) "ACData", "Gear", "WtCGcanvas", and "StrokeDistance" are the visual components of the client GUI and will be discussed in the next section.

The JApplet class "StaticGear" (Pt. 39) is the main program on the client side of the GASP Utility. Once started, all instance data is declared and initialized as needed. The applet then listens for the user to select an aircraft model using a drop box selection object in Panel 1. When the aircraft model is selected, the client contacts the server using RMI and sends a string to identify the selected aircraft. The server returns a HashMap object containing all required initial data for the selected aircraft. The client then uses reflection to efficiently initialize all text fields, text field listeners and most variables in the client applet. The reflection API is very powerful because it provides the ability to represent, or reflect, the classes, interfaces, and objects in the current Java Virtual Machine (JVM) [3]. This initialization process is one of the most important in the utility and warrants a detailed review of the appropriate source code segment as provided below.

--- Start Source Code Segment ---
private void setAircraftValues(
    Gear cn, Gear cp, Gear cs,
    ACdata ca, StrokeDist cd, WtCGcanvas cw,
    HashMap map) throws Exception
{
    Class ccg = cn.getClass();
    Class cca = ca.getClass();
    Class ccd = cd.getClass();
    Class ccw = cw.getClass();
    Set s = map.keySet();
    Iterator i = s.iterator();
    while(i.hasNext())
    {
        // continue here
    }
--- End Source Code Segment ---
09:     String fieldName = (String)i.next();
10:     double value = ((Double)map.get( fieldName )).doubleValue();
11:     if(fieldName.startsWith( "n." ) ) // nose gear
12:         Field f = ccg.getField( fieldName.substring( 2 ) );
13:         f.setDouble( cn, value );
14:         f = ccg.getField( "t" + fieldName.substring( 2 ) );
15:         JTextField jtf = (JTextField)f.get( cn );
16:         jtf.setText( twodig.format( value ) );
17:         jtf.setHorizontalAlignment( javax.swing.SwingConstants.RIGHT);
18:         if( firstAircraft ) jtf.addActionListener(this );
19:     } else if(fieldName.startsWith( "p." ) )
20:         } else if(fieldName.startsWith( "s." ) )
21:         } else if(fieldName.startsWith( "a." ) )
22:         } else if(fieldName.startsWith( "d." ) )
23:         } else if(fieldName.startsWith( "w." ) )
24:         } else // anything else
25:     }
26: 
27:     End Source Code Segment

The method "setAircraftValues" is called when an aircraft model is selected.

Line 1 provides the method statement. The GUI component classes and the aircraft
data HashMap (See Appendix A as example HashMap contents) are passed to the
method. Note that the GUI class descriptions are covered in the next section. Lines
2 through 5 get the "Class" definition of the GUI component classes. Next the
HashMap keyset is obtained in line 6 then used to setup a while loop in lines 7 and
8. Line 9 sets a string variable "fieldName" to the value of the current key in the
HashMap. Line 10 gets and sets the value associated with that key into variable
"value". Next we check to see what class the current key is related to. The prefix
on the key name identifies the class instance (example: n = nose gear instance of
class Gear). If the prefix is "n", then the statements within the Line 11 block if
statement will be executed. Line 12 gets the field (variable) from the class with the same name as the current key (less the two character prefix). Line 13 sets the variable equal to the value associated with the key. Lines 14 through 17 setup the text field associated with the key and value. If this method is being executed for the first time, then line 18 will add a listener to the current text field. Line 19 through 23 carry out the same procedure on the other GUI component classes identified using the key prefix.

**Figure 12: Initial Appearance of Utility**

![Image of GUI components](image)

**Client GUI**

The initial appearance of the Applet GUI once an aircraft model is loaded is presented as Figure 12 above. As indicated by the figure, all input values have been initialized. Also an arbitrary initial aircraft weight and CG (+ symbol near Pt. 40) has been set providing an initial set of results. A detailed description of the GUI component input and results tabs is provided in the next section.
GUI Panel/Tab Descriptions

The following eight figures present a detailed description of each panel or tab available in the GUI. The circled number on each figure below identifies the tab location on the initial GUI view presented in Figure 12.

Figure 13: Aircraft Zero Fuel Gross Weight Envelope Panel

Figure 13 presents an instantiation (w on Figure 11) of class WtCGcanvas which extends JPanel. The purpose of the panel is to graphically present the zero fuel gross weight envelope of the selected aircraft. The envelope provides the weight and center of gravity (CG) limits within which the outfitted aircraft including people must fall before any fuel is added. The envelope is different for each aircraft and is automatically updated and displayed when an aircraft is selected. An initial weight and CG is marked with a blue plus (+) symbol on the envelope. Using the mouse (click, drag or release), any weight or CG value can be selected. When a point is selected by releasing the mouse, the values are passed to the calculation algorithm to update all results as defined in Chapter IV. Note that mass data may also be entered using text fields.
Figure 14: AC Data Tab

Figure 14 presents the "AC Data" (AirCraft Data) tab from instantiation (a on Figure 11) of class ACData which extends JPanel. The purpose of this tab is to present a summary of the aircraft mass properties. The panel is constructed of JLabels, JTextFields, a Jslider and a JComboBox. The aircraft model is selected using the combo box identified "Aircraft Model" (Example = GV). The slider determines the fuel mass added to the aircraft configuration. The top two text fields (labels = ZFGW(lbs) & ZFCG(%MAC)) display the zero fuel weight and CG graphically selected by the user. The next two text fields (labels = Fuel Weight (lbs) & Fuel CG (in)) display the fuel weight selected on the slider and the corresponding fuel longitudinal CG. The last three text fields (labels = Gross Weight(lbs), Xcg(%MAC) & Xcg(in)) present the resulting total aircraft mass properties. Note that the CG is presented in two units, inches and percent of the Mean Aerodynamic Chord (MAC). The MAC is an average chord length located near the center of the wing. The white text fields can be user modified while the gray text fields are for presenting results and are not user editable.
Figure 15: Parameters Tab

Figure 15 presents the "Parameters" tab from instantiation (a on Figure 11) of class ACData. The purpose of this tab is to present any parameters that apply to the entire aircraft other than the mass properties. The panel is constructed of JLabels and JTextFields. The top three text fields (labels = Nx(g), Ny(g) & Nz(g)) of the left column display the load factors applied to the aircraft. The next two fields (labels = mu & Thrust) display the coefficient of friction (mu) for the tires and the total aircraft engine thrust. The last two fields (labels = ZFGW ZCG & Fuel ZCG) on the left column provide nominal values for the zero fuel aircraft and fuel vertical CGs. In the right column, the top four fields (labels = Service Temp - Oil Therm) provide temperatures and pressures that apply to the entire aircraft. The next two fields (labels = Deck Angle & Roll Angle) are the resulting aircraft deck and roll angles from the current aircraft state. The last two fields (labels = MaxMStroke & MaxNStroke) present the maximum strokes for the nose and main landing gear. The white text fields can be user modified while the gray text fields are for presenting results and are not user editable.
Figure 16: Results Tab

Figure 16 presents the "Results" tab from instantiation (n, p, or s on Figure 11) of class Gear which extends JPanel. The landing gear panel has three instantiations of class Gear, one for each landing gear, nose (n), port (p) and starboard (s). The panel presents all data and results related to a landing gear. The results tab is constructed of JLabels and JTextFields. The first two rows of text fields (labels = Axle & Ground) present the coordinates of the landing gear axle and ground contact point of the tire in the body (ACS) coordinate systems. The next two rows (labels = Force(g) & Force(b)) present the landing gear loads in both the ground (GCS) and body (ACS) coordinate systems. The last row provides some option fields for future implementations. The white text fields can be user modified while the gray text fields are for presenting results and are not user editable.
Figure 17: Geometry Tab

The "Geometry" tab from instantiation (n, p, or s on Figure 11) of class Gear which extends JPanel. The landing gear panel has three instantiations of class Gear, one for each landing gear, nose (n), port (p) and starboard (s). The panel presents all data and results related to a landing gear. The geometry tab is constructed of JLabels and JTextFields. This tab presents the geometry required for the landing gear kinematics including the axle, upper reference, lower reference and the pivot. The last row provides some option fields for future implementations. All text fields on this tab are user editable.
Figure 18: Service Tab

<table>
<thead>
<tr>
<th>Results</th>
<th>Geometry</th>
<th>Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>Air Pressure</td>
<td>207.00</td>
<td>Option 6</td>
</tr>
<tr>
<td>Air Area</td>
<td>7.05</td>
<td>Oil Area</td>
</tr>
<tr>
<td>Air Volume</td>
<td>90.30</td>
<td>Oil Volume</td>
</tr>
<tr>
<td>Tire Radius</td>
<td>10.50</td>
<td>Option 7</td>
</tr>
<tr>
<td>Tire Stiffness</td>
<td>6900.00</td>
<td>Option 8</td>
</tr>
<tr>
<td>Option 5</td>
<td>0.00</td>
<td>Option 9</td>
</tr>
</tbody>
</table>

Figure 18 presents the "Service" tab from instantiation (n, p, or s on Figure 11) of class Gear which extends JPanel. The landing gear panel has three instantiations of class Gear, one for each landing gear, nose (n), port (p) and starboard (s). The panel presents all data and results related to a landing gear. The service tab is constructed of JLabels and JTextFields. This tab presents the service data for a given landing gear. The data includes air and oil areas and volumes, tire radius and stiffness and air pressure. Several option fields are available for future implementations. All text fields on this tab are user editable.
Figure 19: Gear Stroke Tab

Figure 19 presents the "Gear Stroke" tab from instantiation (d on Figure 11) of class StrokeDistance. The purpose of this tab is to present the landing gear stroke information. The panel is constructed of JLabels, JTextFields and JSliders. The top three text fields provide the landing gear stroke in inches for each gear as labeled. The next three fields provide the same landing gear stroke in terms of percent of the maximum stroke provided in Figure 15. The three slider bars on the bottom of the tab again present the current stroke in terms of percent of the maximum stroke. All text field and sliders presented on this tab are updated in real time. Also, the landing gear stroke information presented on this tab is all calculated and therefore not user editable.
Figure 20: Distance to Ground Tab

<table>
<thead>
<tr>
<th>Gear Stroke</th>
<th>Distance to Ground</th>
</tr>
</thead>
<tbody>
<tr>
<td>Winglet Tip</td>
<td>797.12 560.57 132.68</td>
</tr>
<tr>
<td>VT Tip</td>
<td>1013.50 0.00 260.19</td>
</tr>
<tr>
<td>Lower Win...</td>
<td>712.56 531.00 57.20</td>
</tr>
<tr>
<td>Selected</td>
<td>0.00 0.00 0.00</td>
</tr>
</tbody>
</table>

Ground to Winglet Tip 154.10

to VT Tip 308.10
to Lower Wing Tip 76.10
to Selected -2.64

Figure 20 presents the "Distance to Ground" tab from instantiation (d on Figure 11) of class StrokeDistance. The purpose of this tab is to present ground clearance information at several locations on the aircraft. The panel is constructed of JLabels and JTextFields. The first three rows of text fields (labels = Winglet Tip, VT Tip & Lower Wing Tip) provide the coordinates (X, Y, and Z) in the aircraft system (ACS) of points for which the ground clearance may be needed. The fourth row of three text fields (label = Selected) is initialized to zero so the user may input any point of interest. The bottom four text fields present the resulting distance to ground for each of the previous points. The four top rows can be user modified while the bottom four fields are for presenting results and are not user editable.
Chapter III
COMMUNICATIONS

The two major communication technologies used in the GASP Utility are the Transmission Control Protocol/Internet Protocol (TCP/IP) and Remote Method Invocation (RMI). These technologies are used to get the applet to the client computer and for the client to communicate with the server to obtain aircraft data. The technologies as they relate to the GASP Utility are discussed below.

**Figure 21: Communication in the GASP Utility**
TCP/IP

The Transmission Control Protocol/Internet Protocol (TCP/IP) is the most popular family of protocols in use today. A protocol in the context of computers and networking is just an agreed upon set of rules for communicating. TCP/IP is not vendor specific and has been implemented on almost everything including Local Area Networks (LANs) and the WWW. The family of TCP/IP protocols can be divided into four major layers. The layers are the Application, Transport, Internet and the Network Access Layers as seen in Figure 22 [1]. The protocols or services provided with TCP/IP that are used by the GASP are discussed below.

Figure 22: TCP/IP Layers

<table>
<thead>
<tr>
<th>Layer</th>
<th>Protocols</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Application Layer = [ User Process ] [ User Process ]</td>
</tr>
<tr>
<td>3</td>
<td>Transport Layer = [ TCP ] [ UDP ]</td>
</tr>
<tr>
<td>2</td>
<td>Internet Layer = [ ICMP ] [ IP ] [ ARP ] [ RARP ]</td>
</tr>
<tr>
<td>1</td>
<td>Network Access Layer = [ Hardware Interface ]</td>
</tr>
</tbody>
</table>

Layer 1 - Network Access Layer

The lowest level of the TCP/IP suite is the Network Access Layer. Since networks must impose constraints on data they transmit, this layer is responsible for restructuring data into a form suitable for network transmission. Also, this layer is responsible for mapping logical addresses to physical device addresses [1].

Layer 2 - Internet Layer

The Internet layer provides two important protocols relevant to this project. The Internet Protocol (IP) and the Internet Control Message Protocol (ICMP) (Pt. 48 on Figure 21). These protocols are discussed in detail below.
The Internet Protocol (IP) performs some of the most important networking functions for handling datagrams. A datagram is just a packet of data or information. The IP takes care of the datagram standardization, routing, fragmentation and delegation.

The datagrams have a standardized format and content including a IP message header. The header has five or six 32-bit words that specify the length, the addresses (source and destination) and the higher level TCP/IP protocol that will handle the data. The standardized format of the IP data is what makes it possible to exchange the datagrams over a network through Gateways.

A central purpose of TCP/IP is to allow exchange of data within computer networks. The GASP Utility data exchange between the client (Pt. 49) and the server (Pt. 50) is an example. The device that makes this exchange possible is called a Gateway. At times, many gateways may be required for a datagram to reach a destination (Pt. 51). Typically a network or gateway will impose an upper limit on the size of a transmitted datagram, called the Maximum Transmission Unit (MTU). To overcome this size limitation, the datagram must be fragmented into smaller datagrams and then reassembled when possible. The IP handles datagram routing and fragmentation functions automatically.

The remaining major function of the IP is to delegate or pass the datagram off to a higher level protocol. The datagram header stores a protocol number. Using this number, the IP can determine to what protocol the current datagram should be delivered.

It should be noted that IP is a connectionless or stateless protocol. So, IP could deliver a datagram to the wrong gateway or host. In addition, IP is an unreliable protocol in that it does not verify that data has been transmitted correctly. To handle these functions, IP must rely on the help of higher level protocols [1].
The other key protocol in the Internet layer is the Internet Control Message Protocol (ICMP). This protocol regulates the flow of datagrams by adjusting transmission speed, detecting unreachable destinations, dynamically re-routing datagrams and verifying operation of IP protocols on remote systems. If a network device finds that the datagram transmission rate is too fast, it will send the source host an ICMP message to temporarily stop sending datagrams. Likewise, the source host will receive an ICMP message if it attempts to send data to an unreachable host. Dynamic re-routing and host verification by the ICMP helps ensure a datagram will reach its destination reliably [1].

**Layer 3 - Transport Layer**

The next higher protocol layer in the TCP/IP family is the Transport layer. This layer has two major protocols which are the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP).

The TCP is an inefficient protocol but adds three important functions (Pt. 48). First, it ensures data transmission reliability by error checking and re-transmitting if needed. Second, it re-assembles the datagrams into a continuous stream of data in the proper sequence. Datagrams may not always follow the same route from the source to destination and therefore may not arrive in the proper order. The TCP uses a sequence number stored in each datagram to reconstruct the datagrams into the original sequence or stream. The third TCP function is to ensure that the data is delivered to the proper application program for processing [1].

The User Datagram Protocol (UDP) is a fast but an unreliable and connectionless protocol. The receiving application must verify the data content and sequence. This protocol may be useful if datagrams less than an MTU in size are used.
Layer 4 - Application Layer

The last protocol layer in the TCP/IP family is the Application Layer. This layer includes protocol support for applications that use data delivered by the TCP/IP including electronic mail and the World Wide Web. The protocols in this layer are called services. The power and functionality of these services are one of the main reasons for the popularity of the TCP/IP family today [1]. A few of the major services provided in the Application Layer are listed in Table IV.

Table IV: Example Application Layer Services

<table>
<thead>
<tr>
<th>Service/Protocol Name</th>
<th>Example Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain Name Service (DNS)</td>
<td>IP Address Alias</td>
</tr>
<tr>
<td>File Transfer Protocol (FTP)</td>
<td>File Transfers</td>
</tr>
<tr>
<td>SMTP</td>
<td>Electronic Mail</td>
</tr>
<tr>
<td>POP</td>
<td>Electronic Mail</td>
</tr>
<tr>
<td>Hypertext Transfer Protocol (HTTP)</td>
<td>World Wide Web</td>
</tr>
</tbody>
</table>

The HTTP is a primary protocol service used on the World Wide Web. The HTTP has two important aspects. First, HTTP gives most of the data interpretation responsibility to the client. Therefore, the client side application receiving the data, like a WWW browser, must be capable of working with various data formats and encodings. The GASP Utility applet is an example of one data format handled by the browser (Pt. 52). The HTTP also provides us with a means to uniquely identify an object. This means is called the Universal Resource Locator (URL). The URL address has five components [1]. An example URL is provided in Table V with the five components identified.
Table V: URL Components

<table>
<thead>
<tr>
<th>No.</th>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>http://</td>
<td>protocol type</td>
</tr>
<tr>
<td>2</td>
<td><a href="http://www.cs.gasou.edu">www.cs.gasou.edu</a>:</td>
<td>host name</td>
</tr>
<tr>
<td>3</td>
<td>8080</td>
<td>port number</td>
</tr>
<tr>
<td>4</td>
<td>~/mray/index.html</td>
<td>file system path</td>
</tr>
<tr>
<td>5</td>
<td>#section1</td>
<td>reference to a named anchor inside a file</td>
</tr>
</tbody>
</table>

RMI

The basics of Remote Method Invocation (RMI) were discussed in the introduction. This chapter provides a more detailed review of RMI as it relates to the GASP Utility. RMI is a pure Java technology that builds on the capabilities of an older technology called Remote Procedure Calls (RPC). The major shortcoming of RPC was that it could only work with primitive data types. RMI extends that capability to include objects. The major functions of RMI are to establish the communication between a Java server and client, handle the marshaling and un-marshaling of parameters and return values, and provide a registry server that maintains a database of remote objects [1]. With these attributes, "RMI allows Java objects running on the same or separate computers to communicate with one another via remote method calls. Such method calls appear the same as those operating on objects in the same program [2]."
Marshaling is a principle we need to understand when using the RMI technology. A primitive value or an object must be converted into an ASCII format before it can be transmitted over a network including the Internet. This process is called marshaling. Once the object has traveled over the network, it must be un-marshaled before it can be used by the receiving program. An example of this process is provided in Figure 23. A client process (Pt. 53 on Figure 23) needs to send parameters to a server process (Pt. 54). The parameters (Pt. 55) and the return values (Pt. 56) must be marshaled before they can be sent.
Another aspect of RMI we need to review is the registry. The RMI registry maintains a database of all available remote objects. Figure 24 demonstrates how the registry works. The registry service (Pt. 57 on Figure 24) must be started before the server or client. With the registry running, the server (Pt. 58) is started which registers (Pt. 59) all remote objects with the RMI Registry. Now the objects references are in the registry database and are available for remote access from clients. When a client is started (Pt. 60) and requests a remote object (Pt. 61), the RMI registry will return a reference to that object (Pt. 62). Once the remote object reference is returned, the client may use that object just as if it existed on the local computer [1].

To implement a program using the RMI technology involves four important steps. First, a description of how the server and client will communicate must be defined using a Java remote interface. Second, the implementation of the server that uses the remote interface must be developed. Third, the client applet (or application) that uses the remote interface reference must be developed. The last step is just to compile and execute the server and client. Figure 25 and source code segments below are used to develop these steps.
Server Interface and Implementation

The server side components (Pt. 63 on Figure 25) require an interface and an implementation of that interface. The Java interface for the GASP Utility is short and simple as seen in the code segment below.

```java
import java.rmi.*;
import java.util.*;
public interface ACDataServer extends Remote {
    public HashMap getAircraftInfo(String acType) throws RemoteException;
}
```

The utility only needs one interface "ACDataServer" as seen in line 3 above. The interface must extend "Remote" and define any methods that will be available for remote access. The GASP Utility just has one remote method called "getAircraftInfo" as given in line 5. The method is of type HashMap and receives a
string parameter. The actual source code of the method is provided in their server implementation file "ACDataServerImpl". Note that the methods defined in the interface file must be able to catch a "RemoteException".

Start Source Code Segment
7: public class ACDataServerImpl extends UnicastRemoteObject
   implements ACDataServer
End Source Code Segment

The next step is to define the server implementation that implements the interface and provides the details for the remote methods. The source code segment above starts the server implementation file "ACDataServerImpl" for the GASP Utility. The server implementation that is going to provide the remote objects must extend "UnicastRemoteObject" and implement the server interface as seen in line 7 above.

Start Source Code Segment
8: public static void main( String args[] ) throws Exception
9: {
10:       System.err.println(
11:             "Initializing server: please wait." );
12:       // create server object
13:       ACDataServerImpl aircraft = new ACDataServerImpl();
14:       // bind ACDataServerImpl object to the rmiregistry
15:       String serverObjectName =
16:           "//steed-100.catia.gulfaero.com/DataServer";
17:       Naming.rebind( serverObjectName, aircraft );
18:       System.err.println( "The ACData Server is up and running." );
19:   }
End Source Code Segment

The actual object "aircraft" of type "ACDataServerImpl" is created in the "main" method of the server implementation application of the GASP Utility as seen in line 12 above. Line 13 defines a string "serverObjectName" as the host computer where the object "aircraft" needs the register itself as a remote service. This line indicates that "steed-100.catia.gulfaero.com" is the host for the RMI registry (Pt. 64) service and the name that a client must use to locate the service is
"DataServer". Line 14 actually binds the object "aircraft" to the RMI registry using the name "serverObjectName". The few lines of code above are all that is required to setup the server side communications when using RMI technology. The setup on the client side is discussed next.

**Client RMI Implementation**

The client side implementation (Pt. 65) of RMI requires very little source code to establish communication with the remote objects in the server implementation. The source code segment below presents the steps required using the client implementation of the GASP Utility.

```java
// name of remote server object bound to rmi registry
17: String serverObjectName =
    "//steen-100.catia.gulfaero.com/DataServer";
// lookup ACDataServerImpl remote object in rmi registry
18: ACDataServer mydata = (ACDataServer) Naming.lookup( serverObjectName );
   // Get remote ACInfo object from Server
19: ac = mydata.getAircraftInfo( acType );
```

Line 17 defines a string "serverObjectName" which provides the host location of the RMI Registry and the service name. The service reference is located (Pt. 66 and 67) in the registry using class “Naming” with method “lookup” and assigned to object “mydata” of type “ACDataServer” in line 18. The final step is to call the remote method “getAircraftInfo” in object “mydata” passing a string (Pt. 68) to define the aircraft type. The method returns the aircraft data as a HashMap (Pt. 69) which is stored in HashMap variable "ac" as seen in line 19.

**Compile and Execute**

The procedure to compile and execute the GASP Utility is provided in Chapter V. The same procedure can be applied to any RMI implementation.
Chapter IV

COMPUTATIONAL METHODS

The computation section of the applet is based on the static equilibrium of the aircraft. All Gulfstream jet aircraft have a tricycle gear configuration with a nose landing gear located in the forward fuselage and a main landing gear located on the inboard, aft section of each wing as seen in Figure 26. This chapter first provides a brief description of the landing gear then develops some of the key computations used in the GASP Utility. The current static equilibrium solution is symmetric. Therefore, all port landing gear results are set equivalent to the starboard values. A future upgrade to the GASP Utility will be to incorporate the un-symmetric solution.

Figure 26: Landing Gear Location
Landing Gear Description

The main landing gear is an oleo-pneumatic type with an articulated, trailing arm axle with two 34 inch diameter tires. The nose gear is also an oleo pneumatic strut, but of a cantilever configuration with the axle attached directly to the piston of the strut. The spring force holding up the weight of the aircraft is developed by compressing nitrogen in the cylinder. Figure 27 shows a picture of the main gear (excluding the side brace, wheels, and tires) with the structural member connection/joints locations and types listed.

Figure 27: Main Gear Major Components

Note: Wheels, tires, brakes, and side brace are removed for clarity.
Aircraft Balance

The aircraft can be considered a free body in a basic statics problem with forces similar to those seen in Figure 28. The three landing gear ground contact points are the only constraint locations. The load distribution between the landing gear is dependent on many factors including the aircraft weight, center of gravity (CG), external forces, and the gear compression. The weight and center of gravity of the aircraft can vary drastically depending on how the aircraft is outfitted and on the quantity of fuel carried. The aircraft is also subjected to different external forces such as thrust or induced load factors. Load factor sources can include turning, braking and accelerating. In addition, certain combinations of load factors require evaluation by the Federal Aviation Administration (FAA). The source code segments providing the equations used to calculate the gear loads and some other key results will be defined in the following paragraphs. An alphabetically sorted list of all variables used in the equations is provided in Appendix B for reference. In addition, a listing of all source code for the GASP Utility is provided in Appendix C.

Figure 28: Example Aircraft Loading
Solution Methods

The first step in the solution is to determine the total aircraft mass properties based on user input. The user graphically selects the aircraft zero fuel gross weight and CG, then a total fuel weight is entered using the fuel slider bar. Note that these three parameters can be set directly by entering the values in the corresponding text fields. The fuel longitudinal and vertical CGs are determined by interpolation on data tables using the fuel weight. With the mass components defined, the total properties are simple to calculate using the following equations:

------------ Start Source Code Segment ------------
// Compute the total aircraft mass properties
1: a.gw = a.zfgw + a.fuelw;
2: zfcgi = w.macLE + w.mac / 100.0 * a.zfcg;
3: a.cgi = (( zfcgi * a.zfgw ) + ( a.fuelcg * a.fuelw ) ) / a.gw;
4: a.cgz = (( a.zcgz * a.zfgw ) + ( a.zcgf * a.fuelw ) ) / a.gw;
5: a.cgp = 100.0 * (a.cgi - w.macLE ) / w.mac;
------------ End Source Code Segment ------------

Equation (1) calculates the total aircraft weight. Equation (2) calculates the zero fuel CG in inches. Equations (3) and (4) calculate the longitudinal and vertical total aircraft CG in inches. Equation (5) calculates the longitudinal CG again in terms of the Mean Aerodynamic Chord (MAC).

The next step is to calculate the landing gear loads and aircraft static position taking into account the selected total mass properties and any additional parameters. The landing gear deflection (stroke) is a non-linear function of applied force. Therefore, an iterative solution is required to calculate the gear loads and static position. Since the gear position has only a minor effect on gear forces, the solution converges fast with a loop set to iterate a constant value of six times. The sequence of calculations inside the loop for the symmetric solution is provided below. The landing gear kinematics are calculated using equations 6 through 14.
--- Start Source Code Segment ---

for ( int i=0; i<6; i++ )
{
    // compute nose gear kinematics
    d.nstr = Math.max(0.0, d.nstr);
    n.ax = n.axo + d.nstr * Math.sin(n.theta);
    n.az = n.azo + d.nstr * Math.cos(n.theta);

    // compute main gear kinematics
    d.sstr = Math.max(0.0, d.sstr);
    s.th3 = Math.acos((s.rpl * s.rpl + s.rpu * s.rpu - (s.rul - d.sstr) * (s.rul - d.sstr)) / (2.0 * s.rpl * s.rpu));
    s.thax = s.theta + s.thetaUPL - s.th3;
    s.ax = s.px + s.rpa * Math.cos(s.thax);
    s.az = s.pz + s.rpa * Math.sin(s.thax);
--- End Source Code Segment ---

Equation (6) initiates the nose gear stroke to 0.0 or the current value, whichever is greater. Equations (7) and (8) calculate the longitudinal and vertical axle positions of the nose gear. The main gear calculations are the same as given by equations (9), (13) and (14). Equations (10) and (11) provide additional geometry terms required by the trailing arm type main gear.

--- Start Source Code Segment ---

rnm = Math.sqrt((s.ax - n.ax) * (s.ax - n.ax) + (s.az - n.az) * (s.az - n.az));
alphaax = Math.atan((s.az - n.az) / (s.ax - n.ax));
alphat = Math.asin((n.rt - s.rt) / rnm);
--- End Source Code Segment ---

Next, the distance between the nose and main gear axle locations is calculated with equation (15) and the angle of the line through the axles is computed in equation (16). The angle formed by the different sizes of the nose and main gear tires is computed in equation (17). These values are required to calculate the aircraft deck angle (pitch) which is the angle between the Aircraft Coordinate System (ACS) longitudinal axis and the Ground Coordinate System (GCS) longitudinal axis as seen in Figure 29.
Equations (16) through (18) calculate the aircraft pitch angle. The trig functions of the pitch angle are required several times in the solution. Therefore, to increase the speed of the algorithm, these values are pre-calculated with equations (19) and (20). Now the longitudinal and vertical location of the landing gear tires to ground contact points may be calculated.

The nose and main gear ground contact points are given by equations (21) through (22) and (23) through (24) respectively. Next the main and nose gear loads in the GCS are determined.
Equation (25) computes the main gear vertical load by taking moments about the nose gear ground contact point. The longitudinal main gear loads (26) are just the vertical loads (25) multiplied by a coefficient of friction. Equation (27) provides the vertical loads for the nose gear by summing vertical forces on the aircraft. The nose gear will not have a longitudinal load in the GCS for the symmetric solution since there are no nose gear brakes. The following equations revise the tire radius and gear geometry required for the next iteration.

--- Start Source Code Segment ---

// compute tire radii
28: n.rt = n.rto - n.fzg / n.kt;
29: s.rt = s.rto - s.fzg / s.kt;

// NLG oleo calculations
30: n.fo = n.fzg * Math.cos(n.theta + pitch);
31: n.pp = Math.max(n.fo / n.aa, n.po);
32: d.nstr = n.vo/n.aa * (1.0 - (n.po + a.ampress) / (n.pp + a.ampress));

// MLG oleo calculations
33: s.fo = s.fzg * ((s.rpa * (Math.cos(s.thax - pitch) - a.mu * Math.sin(s.thax - pitch)). + a.mu * s.rt) * (s.rul - d.sstr)) /(s.rpl * s.rpu * Math.sin(s.th3));
34: s.pp = Math.max(s.fo / s.aa, s.po);
35: d.sstr = s.vo/s.aa * (1.0 - (s.po + a.ampress) / (s.pp + a.ampress));
--- End Source Code Segment ---

Equations (28) and (29) define the revised tire radius assuming the tires behave as linear springs. The landing gear strokes are calculated using the ideal gas laws to determine the compression required to give the calculated vertical gear force. The gear stroke is the convergence variable. When the change in gear stroke between iterations is near zero, the solution is converged. As the gear stroke changes, the distance between the nose and main gear will also change due to gear kinematics. However, this distance change (about 10 inches max) is very small relative to the total distance between the gears (about 500 inches). This leads to very small gear load changes with each iteration providing for a fast
convergence. The nose gear stroke is defined by equation (32) using the oleo force and pressure given by equations (30) and (31). The main gear stroke is calculated in a similar fashion with equations (33) through (35) which include the kinematics of the trailing arm configuration. The stroke calculations conclude the current iteration. The results defined within the loop are assumed converged after six cycles. When the loop terminates, the remaining results are calculated.

--- Start Source Code Segment ---

```c
// Set Transformation matrix
36: setTrans1();

// Body Axis Gear Loads
37: doTrans(0.0, 0.0, 0.0, n.fxg, n.fyg, n.fzg);
38: n.fxb = acs[0];
39: n.fyb = acs[1];
40: n.fzb = acs[2];
} //end of loop
--- End Source Code Segment ---
```

The method (36) defines a transformation matrix that rotates results from the GCS to the ACS. Method (37) uses this matrix along with the ground system loads to define the ACS results that are placed in the global array "acs[]". This array is then used to set the nose gear ACS loads variables in equations (38) through (40). The same procedure is followed for the main gear loads. One of the final calculations is the distance to the ground from certain locations on the aircraft.

--- Start Source Code Segment ---

```c
//compute ground height
41: doTrans(n.gx, n.gy, n.gz, d.wltx, d.wlty, d.wltz);
42: d.twltd.setText (twodig.format(grd[2]));
--- End Source Code Segment ---
```

The same type of transformation method used to rotate loads in (37) may also be used to rotate geometry about an arbitrary point. The first three parameters in the method call (41) define the point in ACS to rotate about. The last three parameters are the coordinates of the point to rotate (example: Tip of the winglet). The results of the method call are in the "grd[]" array. Equation (42) provides an
example of how the results may be used to set a text field where grd[2] provides the distance to the ground from the winglet tip. The GASP Utility provides a ground clearance panel (Figure 20) where the distance to the ground is provided for four points. Three points are pre-defined as seen in Figure 30. Any of the points can be redefined by modifying the appropriate text fields.

--- Start Source Code Segment ---

```c
// calculate % stroke
43: d.nstrp = d.nstr / a.mngs * 100.;
44: d.sstrp = d.sstr / a.mmgs * 100.;
--- End Source Code Segment ---
```

One of the last calculations is to find the percent of the maximum landing gear stroke for the nose and main landing gears. Equations (43) and (44) define these values which are used to set the percent stroke text fields and slider bars.

**Figure 30: Ground Clearance**

Vertical tail tip to Ground

Winglet Root and Tip to Ground
Chapter V
USER’S GUIDE

This chapter provides a detailed users guide for the GASP Utility. First, the server side administration is covered including how to start the server and modify baseline aircraft data files. It is assumed that the components of the utility are installed on a host that uses a UNIX operating system. Next, the usage of the client applet is covered. Since the client is an applet, the applet usage is independent of operating system. Contact the Loads and Dynamics Group at Gulfstream Aerospace for information about the current location and access to the source code and data components of the GASP Utility.

Starting the Server

The utility server is a Java application that uses RMI to communicate with client(s). Since RMI is used, the RMI registry must be started before the application server is started. It is required that the registry is started from within the web server’s file system on which the utility is located. The RMI registry should be available on any system that has a Java Runtime Environment (JRE) installed. To start the registry, just type the commands:

```
$ cd xxxxxxxxxx/StaticGear
$ nohup rmiregistry &
```

First go to the StaticGear directory. The rest of the path name "xxxxxxxxxxx" will depend on where the utility is implemented. The "rmiregistry" command is preceded by "nohup" so the registry will not be terminated if the terminal session is closed. The "&" after the command will run the registry in the background. See Chapter III for additional detail on the RMI technology.

The next step is to compile and execute the server. In order to execute
components of the utility, all Java source code must be compiled into Java class files. In addition, the class files which extend the RMI object "UnicastRemoteObject" must be re-compiled using the RMI compiler "rmic". The only class file meeting this requirement in this utility is the server implementation file "ACDataServerImpl.class". Therefore, the following two commands will compile the utility in the required manner:

```bash
$ javac *.java
$ rmic -v1.2 ACDataServerImpl
```

Now the file server "ACDataServerImpl.class" can be started using the command:

```bash
$ nohup java ACDataServerImpl &
```

The command is preceded by "nohup" so the file server will not be terminated if the terminal session is closed. The "&" after the command will run the file server in the background. At this point the RMI registry and file server are running and waiting for client request.

**Baseline Aircraft Data File Modification**

All of the aircraft data files are located in the StaticGear directory. The files are just ordinary text files and may be modified using any text editor. An example file is provided in Appendix A. Note that all aircraft data must be entered in variable name plus variable value pairs. The variable name must be proceeded by the two character prefix that identifies the class instance where the variable is declared in the utility. Comments may be placed in the file on any line that starts with the characters "/\".

**Starting the GASP Utility (Client Applet)**

The client is easy to start using World Wide Web browsers like Netscape Communicator or Microsoft Internet Explorer. The user just needs to navigate to the web page that contains the HTML link to the utility applet. Once the link is
selected, the applet will automatically be loaded and executed in a web browser window. Note that at the time of the utility's release, the "Java 1.3" browser plug-in was required to be installed on most browsers for the utility to execute properly. The plug-in is available from many Java support web pages.

**Figure 31: Initial Appearance of Utility**

Selecting and Loading Aircraft Data

The initial appearance of the Applet GUI once an aircraft model is loaded is presented as Figure 31 above. The aircraft model is selected by using the combo box selection tool identified in Figure 31 by (Pt. 70). Place the mouse on the combo box and click and hold. All available aircraft models will be displayed. Still holding the mouse, move to the desired aircraft model and release the mouse. The aircraft and results will be initiated using the selected aircraft model data. The user may reset the current data to the initial conditions at any time by re-selecting the current
aircraft model. In addition, a new aircraft model can be selected at any time.

**Modifying Data from within the Utility**

After the applet has been initialized using data from an aircraft model file, the user has several methods available to modify the data. The user may graphically select the aircraft's zero fuel gross weight and center of gravity by dragging or clicking the mouse anywhere on the mass properties canvas (Pt. 71). Fuel is added to the aircraft using the vertical slider (Pt. 72) located in the aircraft data panel. The user may also update any aircraft information by modifying the editable text fields. Any white text field may be modified at any time (Pt. 73). The gray text fields are for presenting results and are not user editable (Pt. 74). All results are updated dynamically as the mass properties or other parameters are modified.

**Utility Termination**

To terminate the utility, just close the browser window in which the utility is running (example = Pt. 75). The applet will stop all utility processes and close network connections as required.
Chapter VI
SUMMARY

The GASP Utility as defined in this report has been implemented successfully within the Gulfstream Aerospace Engineering environment. The utility has met or exceeded all design objectives as outlined in Table II. The Gulfstream Loads and Dynamics Group plans to evolve the utility by extending the symmetric solution to a full un-symmetric solution. In addition, several other applications within the Gulfstream Engineering organization have been identified for potential re-implementation using the technologies and lessons learned while developing the GASP Utility.

Contact the Loads and Dynamics Group at Gulfstream Aerospace Corporation for information about the current location and access to the source code and data components of the GASP Utility.
REFERENCES


3. Sun Micro Systems Tutorial World Wide Web Page. The site URL is as follows:
   http://java.sun.com/docs/books/tutorial/index.html
APPENDICES
**APPENDIX A**

**SAMPLE INPUT DATA**

Table VI: GVdata.txt

<table>
<thead>
<tr>
<th>Aircraft Data for the GV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aircraft weight-CG data</td>
</tr>
<tr>
<td>w.mac</td>
</tr>
<tr>
<td>w.macLE</td>
</tr>
<tr>
<td>w.offsetCG</td>
</tr>
<tr>
<td>w.offsetWT</td>
</tr>
<tr>
<td>w.deltaCG</td>
</tr>
<tr>
<td>w.deltaWT</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Zero Fuel Envelope CG Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>w.zfCG1</td>
</tr>
<tr>
<td>w.zfCG2</td>
</tr>
<tr>
<td>w.zfCG3</td>
</tr>
<tr>
<td>w.zfCG4</td>
</tr>
<tr>
<td>w.zfCG5</td>
</tr>
<tr>
<td>w.zfCG6</td>
</tr>
<tr>
<td>w.zfCG7</td>
</tr>
<tr>
<td>w.zfCG8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Zero Fuel Envelope Weight Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>w.zfWT1</td>
</tr>
<tr>
<td>w.zfWT2</td>
</tr>
<tr>
<td>w.zfWT3</td>
</tr>
<tr>
<td>w.zfWT4</td>
</tr>
<tr>
<td>w.zfWT5</td>
</tr>
<tr>
<td>w.zfWT6</td>
</tr>
<tr>
<td>w.zfWT7</td>
</tr>
<tr>
<td>w.zfWT8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Coefs for Fuel Burn Curve CG(WT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>w.fCG1</td>
</tr>
<tr>
<td>w.fCG2</td>
</tr>
<tr>
<td>w.fCG3</td>
</tr>
<tr>
<td>w.fCG4</td>
</tr>
<tr>
<td>w.fCG5</td>
</tr>
<tr>
<td>w.fCG6</td>
</tr>
<tr>
<td>w.fCG7</td>
</tr>
<tr>
<td>w.fCG8</td>
</tr>
<tr>
<td>w.fCG9</td>
</tr>
<tr>
<td>w.fWT1</td>
</tr>
<tr>
<td>w.fWT2</td>
</tr>
<tr>
<td>w.fWT3</td>
</tr>
<tr>
<td>w.fWT4</td>
</tr>
<tr>
<td>w.fWT5</td>
</tr>
<tr>
<td>w.fWT6</td>
</tr>
<tr>
<td>w.fWT7</td>
</tr>
<tr>
<td>w.fWT8</td>
</tr>
<tr>
<td>w.fWT9</td>
</tr>
</tbody>
</table>

// Geometry Data
// Nose Geometry

| n.axo | 86.830 |
| n.ayo | 0.000 |
| n.azo | 0.220 |
| n.uxo | 93.870 |
| n.uyo | 0.000 |
| n.uzo | 49.570 |
| n.lxo | 85.660 |
| n.lyo | 0.000 |
| n.lzo | 0.400 |
| n.px  | 0.000 |
| n.py  | 0.000 |
| n.pz  | 0.000 |
| n.opt3| 0.000 |
| n.opt4| 0.000 |

// Port Main Geometry

| p.axo | 617.067 |
| p.ayo | -86.000 |
| p.azo | -16.415 |
| p.uxo | 622.920 |
| p.uyo | -86.000 |
| p.uzo | 41.310 |
| p.lxo | 619.441 |
| p.lyo | -86.000 |
| p.lzo | -12.266 |
Table VI: GVdata.txt (Continued)

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>p.px</td>
<td>603.724</td>
<td></td>
</tr>
<tr>
<td>p.py</td>
<td>-86.000</td>
<td></td>
</tr>
<tr>
<td>p.pz</td>
<td>4.132</td>
<td></td>
</tr>
<tr>
<td>p.opt3</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>p.opt4</td>
<td>0.000</td>
<td></td>
</tr>
</tbody>
</table>

// Star Main Geometry

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>s.axo</td>
<td>617.067</td>
<td></td>
</tr>
<tr>
<td>s.ayo</td>
<td>86.000</td>
<td></td>
</tr>
<tr>
<td>s.azo</td>
<td>-16.415</td>
<td></td>
</tr>
<tr>
<td>s.uxo</td>
<td>622.920</td>
<td></td>
</tr>
<tr>
<td>s.uyo</td>
<td>86.000</td>
<td></td>
</tr>
<tr>
<td>s.uzo</td>
<td>41.310</td>
<td></td>
</tr>
<tr>
<td>s.lxo</td>
<td>619.441</td>
<td></td>
</tr>
<tr>
<td>s.lyo</td>
<td>86.000</td>
<td></td>
</tr>
<tr>
<td>s.lzo</td>
<td>-12.266</td>
<td></td>
</tr>
<tr>
<td>s.px</td>
<td>603.724</td>
<td></td>
</tr>
<tr>
<td>s.py</td>
<td>86.000</td>
<td></td>
</tr>
<tr>
<td>s.pz</td>
<td>4.132</td>
<td></td>
</tr>
<tr>
<td>s.opt3</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>s.opt4</td>
<td>0.000</td>
<td></td>
</tr>
</tbody>
</table>

// Service Dat

// Nose Service

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>n.po</td>
<td>207.000</td>
<td></td>
</tr>
<tr>
<td>n.opt5</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>n.aa</td>
<td>7.050</td>
<td></td>
</tr>
<tr>
<td>n.oa</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>n.vo</td>
<td>90.300</td>
<td></td>
</tr>
<tr>
<td>n.ov</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>n.rto</td>
<td>10.500</td>
<td></td>
</tr>
<tr>
<td>n.rt</td>
<td>10.500</td>
<td></td>
</tr>
<tr>
<td>n.opt7</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>n.kt</td>
<td>6900.000</td>
<td></td>
</tr>
<tr>
<td>n.opt8</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>n.opt5</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>n.opt9</td>
<td>0.000</td>
<td></td>
</tr>
</tbody>
</table>

// Port Main Service

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>p.po</td>
<td>250.000</td>
<td></td>
</tr>
<tr>
<td>p.opt6</td>
<td>0.000</td>
<td></td>
</tr>
</tbody>
</table>
Table VI: GVdata.txt (Continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>p.aa</td>
<td>17.683</td>
</tr>
<tr>
<td>p.oa</td>
<td>0.000</td>
</tr>
<tr>
<td>p.vo</td>
<td>362.500</td>
</tr>
<tr>
<td>p.ov</td>
<td>0.000</td>
</tr>
<tr>
<td>p.rto</td>
<td>18.000</td>
</tr>
<tr>
<td>p.opt7</td>
<td>0.000</td>
</tr>
<tr>
<td>p.kt</td>
<td>19200.000</td>
</tr>
<tr>
<td>p.opt8</td>
<td>0.000</td>
</tr>
<tr>
<td>p.opt5</td>
<td>0.000</td>
</tr>
<tr>
<td>p.opt9</td>
<td>0.000</td>
</tr>
<tr>
<td>// Star Main Service</td>
<td></td>
</tr>
<tr>
<td>s.po</td>
<td>250.000</td>
</tr>
<tr>
<td>s.opt6</td>
<td>0.000</td>
</tr>
<tr>
<td>s.aa</td>
<td>17.683</td>
</tr>
<tr>
<td>s.oa</td>
<td>0.000</td>
</tr>
<tr>
<td>s.vo</td>
<td>362.500</td>
</tr>
<tr>
<td>s.ov</td>
<td>0.000</td>
</tr>
<tr>
<td>s.rto</td>
<td>18.000</td>
</tr>
<tr>
<td>s.opt7</td>
<td>0.000</td>
</tr>
<tr>
<td>s.kt</td>
<td>19200.000</td>
</tr>
<tr>
<td>s.opt8</td>
<td>0.000</td>
</tr>
<tr>
<td>s.opt5</td>
<td>0.000</td>
</tr>
<tr>
<td>s.opt9</td>
<td>0.000</td>
</tr>
<tr>
<td>// Initial Gear Results</td>
<td></td>
</tr>
<tr>
<td>// Nose Results</td>
<td></td>
</tr>
<tr>
<td>n.ax</td>
<td>0.000</td>
</tr>
<tr>
<td>n.ay</td>
<td>0.000</td>
</tr>
<tr>
<td>n.az</td>
<td>0.000</td>
</tr>
<tr>
<td>n.gx</td>
<td>0.000</td>
</tr>
<tr>
<td>n.gy</td>
<td>0.000</td>
</tr>
<tr>
<td>n.gz</td>
<td>0.000</td>
</tr>
<tr>
<td>n.fxg</td>
<td>0.000</td>
</tr>
<tr>
<td>n.fyg</td>
<td>0.000</td>
</tr>
<tr>
<td>n.fzg</td>
<td>0.000</td>
</tr>
<tr>
<td>n.fxb</td>
<td>0.000</td>
</tr>
<tr>
<td>n.fyb</td>
<td>0.000</td>
</tr>
</tbody>
</table>
### Table VI: GVdata.txt (Continued)

<table>
<thead>
<tr>
<th>Port Main Results</th>
<th>Star Main Results</th>
<th>Initial A/C Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>p.ax</td>
<td>s.ax</td>
<td>a.zfeg</td>
</tr>
<tr>
<td>p.ay</td>
<td>s.ay</td>
<td>a.zfgw</td>
</tr>
<tr>
<td>p.az</td>
<td>s.az</td>
<td>a.fuelw</td>
</tr>
<tr>
<td>p.gx</td>
<td>s.gx</td>
<td>a.fuelcg</td>
</tr>
<tr>
<td>p.gy</td>
<td>s.gy</td>
<td></td>
</tr>
<tr>
<td>p.gz</td>
<td>s.gz</td>
<td></td>
</tr>
<tr>
<td>p.fxg</td>
<td>s.fxg</td>
<td></td>
</tr>
<tr>
<td>p.fyg</td>
<td>s.fyg</td>
<td></td>
</tr>
<tr>
<td>p.fzg</td>
<td>s.fzg</td>
<td></td>
</tr>
<tr>
<td>p.fxg</td>
<td>s.fxg</td>
<td></td>
</tr>
<tr>
<td>p.fyb</td>
<td>s.fyb</td>
<td></td>
</tr>
<tr>
<td>p.fzb</td>
<td>s.fzb</td>
<td></td>
</tr>
<tr>
<td>p.opt1</td>
<td>s.opt1</td>
<td>a.zfeg</td>
</tr>
<tr>
<td>p.opt2</td>
<td>s.opt2</td>
<td>a.zfgw</td>
</tr>
</tbody>
</table>

| n.fzb             | 0.000             |                  |
| n.opt1            | 0.000             |                  |
| n.opt2            | 0.000             |                  |

<table>
<thead>
<tr>
<th>Initial A/C Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>a.zfeg</td>
</tr>
<tr>
<td>a.zfgw</td>
</tr>
<tr>
<td>a.fuelw</td>
</tr>
<tr>
<td>a.fuelcg</td>
</tr>
</tbody>
</table>
### Table VI: GVdata.txt (Continued)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a.gw</td>
<td>0.000</td>
</tr>
<tr>
<td>a.cgpe</td>
<td>0.000</td>
</tr>
<tr>
<td>a.cgii</td>
<td>0.000</td>
</tr>
<tr>
<td>/a.cgzi</td>
<td>0.000</td>
</tr>
<tr>
<td>a.nx</td>
<td>0.000</td>
</tr>
<tr>
<td>a.ny</td>
<td>0.000</td>
</tr>
<tr>
<td>a.nz</td>
<td>1.000</td>
</tr>
<tr>
<td>a.mu</td>
<td>0.200</td>
</tr>
<tr>
<td>a.thrust</td>
<td>0.000</td>
</tr>
<tr>
<td>a.opt10</td>
<td>0.000</td>
</tr>
<tr>
<td>a.stemp</td>
<td>0.000</td>
</tr>
<tr>
<td>a.temp</td>
<td>0.000</td>
</tr>
<tr>
<td>a.ampress</td>
<td>15.000</td>
</tr>
<tr>
<td>a.oiltc</td>
<td>0.000</td>
</tr>
<tr>
<td>a.decka</td>
<td>0.000</td>
</tr>
<tr>
<td>a.rolla</td>
<td>0.000</td>
</tr>
<tr>
<td>a.zcgz</td>
<td>96.500</td>
</tr>
<tr>
<td>a.zcgf</td>
<td>50.000</td>
</tr>
<tr>
<td>a.mmgs</td>
<td>20.000</td>
</tr>
<tr>
<td>a.mngs</td>
<td>12.160</td>
</tr>
</tbody>
</table>

// Initial Stroke and Distance Data

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>d.pstr</td>
<td>0.000</td>
</tr>
<tr>
<td>d.pstrp</td>
<td>0.000</td>
</tr>
<tr>
<td>d.nstr</td>
<td>0.000</td>
</tr>
<tr>
<td>d.nstrp</td>
<td>0.000</td>
</tr>
<tr>
<td>d.sstr</td>
<td>0.000</td>
</tr>
<tr>
<td>d.sstrp</td>
<td>0.000</td>
</tr>
<tr>
<td>d.wltx</td>
<td>0.000</td>
</tr>
<tr>
<td>d.wlty</td>
<td>0.000</td>
</tr>
<tr>
<td>d.wltz</td>
<td>0.000</td>
</tr>
<tr>
<td>d.vttx</td>
<td>0.000</td>
</tr>
<tr>
<td>d.vtty</td>
<td>0.000</td>
</tr>
<tr>
<td>d.vttz</td>
<td>0.000</td>
</tr>
<tr>
<td>d.rdx</td>
<td>0.000</td>
</tr>
<tr>
<td>d.rdy</td>
<td>0.000</td>
</tr>
<tr>
<td>d.rdz</td>
<td>0.000</td>
</tr>
<tr>
<td>d.selx</td>
<td>0.000</td>
</tr>
<tr>
<td>d.sely</td>
<td>0.000</td>
</tr>
</tbody>
</table>
Table VI: GVdata.txt (Continued)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>d.selz</td>
<td>0.000</td>
</tr>
<tr>
<td>d.wltd</td>
<td>0.000</td>
</tr>
<tr>
<td>d.vltd</td>
<td>0.000</td>
</tr>
<tr>
<td>d.rdd</td>
<td>0.000</td>
</tr>
<tr>
<td>d.seld</td>
<td>0.000</td>
</tr>
<tr>
<td>end</td>
<td></td>
</tr>
</tbody>
</table>
### APPENDIX B

**SOURCE CODE VARIABLE DESCRIPTIONS**

**Table VII: Source Code Variable Descriptions**

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a.ampress</td>
<td>Ambient Pressure</td>
</tr>
<tr>
<td>a.cgi</td>
<td>Aircraft Axial CG (in)</td>
</tr>
<tr>
<td>a.cgp</td>
<td>Aircraft Axial CG (%MAC)</td>
</tr>
<tr>
<td>a.cgz</td>
<td>Aircraft Vertical CG (in)</td>
</tr>
<tr>
<td>a.decka</td>
<td>Deck Angle</td>
</tr>
<tr>
<td>a.fuelcg</td>
<td>Fuel CG</td>
</tr>
<tr>
<td>a.fuelw</td>
<td>Fuel Weight</td>
</tr>
<tr>
<td>a.gw</td>
<td>Aircraft Gross Weight</td>
</tr>
<tr>
<td>a.mngs</td>
<td>Max Main Gear Stroke</td>
</tr>
<tr>
<td>a.mngs</td>
<td>Max Nose Gear Stroke</td>
</tr>
<tr>
<td>a.mu</td>
<td>Drag Coefficient</td>
</tr>
<tr>
<td>a.nx</td>
<td>Aircraft Axial Load Factor</td>
</tr>
<tr>
<td>a.ny</td>
<td>Aircraft Lateral Load Factor</td>
</tr>
<tr>
<td>a.nz</td>
<td>Aircraft Vertical Load Factor</td>
</tr>
<tr>
<td>a.oiltc</td>
<td>Oil Temperature Coefficient</td>
</tr>
<tr>
<td>a.olla</td>
<td>Roll Angle</td>
</tr>
<tr>
<td>a.stemp</td>
<td>Service Temperature</td>
</tr>
<tr>
<td>a.temp</td>
<td>Ambient Temperature</td>
</tr>
<tr>
<td>a.thrust</td>
<td>Engine Thrust</td>
</tr>
<tr>
<td>a.zcgf</td>
<td>Vertical CG of Fuel</td>
</tr>
<tr>
<td>a.zcgz</td>
<td>Vertical CG of ZFGW A/C</td>
</tr>
<tr>
<td>a.zfcg</td>
<td>Initial Zero Fuel Gross CG</td>
</tr>
<tr>
<td>a.zfgw</td>
<td>Initial Zero Fuel Gross Weight</td>
</tr>
<tr>
<td>ac</td>
<td>HashMap Containing Aircraft Data</td>
</tr>
<tr>
<td>alphaax</td>
<td>Angle between Axles and Body Axis</td>
</tr>
<tr>
<td>alphat</td>
<td>Angle between Axles and Ground Axis</td>
</tr>
<tr>
<td>cg</td>
<td>Aircraft Center of Gravity</td>
</tr>
<tr>
<td>cosalpha</td>
<td>Cosine of the Pitch Angle</td>
</tr>
<tr>
<td>d.nstr</td>
<td>Nose Main Gear Stroke</td>
</tr>
<tr>
<td>d.nztrp</td>
<td>Percent of Max Nose Gear Stoke</td>
</tr>
<tr>
<td>d.pstr</td>
<td>Port Main Gear Stroke</td>
</tr>
<tr>
<td>d.pstrp</td>
<td>Percent of Max Port Gear Stoke</td>
</tr>
</tbody>
</table>
### Table VII: Source Code Variable Descriptions (Continued)

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>d.rdd</td>
<td>Wing Tip Ground Clearance</td>
</tr>
<tr>
<td>d.rdx</td>
<td>Initial FS of the Wing Tip</td>
</tr>
<tr>
<td>d.rdy</td>
<td>Initial BL of the Wing Tip</td>
</tr>
<tr>
<td>d.rdz</td>
<td>Initial WL of the Wing Tip</td>
</tr>
<tr>
<td>d.seld</td>
<td>User Selected Ground Clearance</td>
</tr>
<tr>
<td>d.selx</td>
<td>Initial FS of Selected Point</td>
</tr>
<tr>
<td>d.sely</td>
<td>Initial BL of Selected Point</td>
</tr>
<tr>
<td>d.selz</td>
<td>Initial WL of Selected Point</td>
</tr>
<tr>
<td>d.sstr</td>
<td>Starboard Main Gear Stroke</td>
</tr>
<tr>
<td>d.sstrp</td>
<td>Percent of Max Starboard Gear Stroke</td>
</tr>
<tr>
<td>d.vttdd</td>
<td>VT Tip Ground Clearance</td>
</tr>
<tr>
<td>d.vtttx</td>
<td>Initial FS of the VT Tip</td>
</tr>
<tr>
<td>d.vtty</td>
<td>Initial BL of the VT Tip</td>
</tr>
<tr>
<td>d.vttz</td>
<td>Initial WL of the VT Tip</td>
</tr>
<tr>
<td>d.wltd</td>
<td>Winglet Tip Ground Clearance</td>
</tr>
<tr>
<td>d.wltx</td>
<td>Initial FS of the Winglet Tip</td>
</tr>
<tr>
<td>d.wlty</td>
<td>Initial BL of the Winglet Tip</td>
</tr>
<tr>
<td>d.wltz</td>
<td>Initial WL of the Winglet Tip</td>
</tr>
<tr>
<td>firstAircraft</td>
<td>Boolean to Check for First Aircraft</td>
</tr>
<tr>
<td>fuelburnCG[]</td>
<td>Zero Fuel Envelope CG</td>
</tr>
<tr>
<td>fuelburnWT[]</td>
<td>Zero Fuel Envelope Weight</td>
</tr>
<tr>
<td>geoOut[]</td>
<td>Output from the Transformation Method</td>
</tr>
<tr>
<td>ifu</td>
<td>Integer Fuel Quantity (Fuel Slider Output)</td>
</tr>
<tr>
<td>n.aa</td>
<td>Nose Gear Air Area</td>
</tr>
<tr>
<td>n.ax</td>
<td>Nose Gear Axle FS Position</td>
</tr>
<tr>
<td>n.axo</td>
<td>Nose Gear Axle Initial FS Position</td>
</tr>
<tr>
<td>n.ay</td>
<td>Nose Gear Axle FS Position</td>
</tr>
<tr>
<td>n.ayo</td>
<td>Nose Gear Axle Initial BL Position</td>
</tr>
<tr>
<td>n.az</td>
<td>Nose Gear Axle FS Position</td>
</tr>
<tr>
<td>n.azo</td>
<td>Nose Gear Axle Initial WL Position</td>
</tr>
<tr>
<td>n.fxb</td>
<td>Nose Gear Axial Loads (ACS)</td>
</tr>
<tr>
<td>n.fxg</td>
<td>Nose Gear Axial Loads (GCS)</td>
</tr>
<tr>
<td>n.fyb</td>
<td>Nose Gear Lateral Loads (ACS)</td>
</tr>
<tr>
<td>n.fyg</td>
<td>Nose Gear Lateral Loads (GCS)</td>
</tr>
<tr>
<td>n.fzb</td>
<td>Nose Gear Vertical Loads (ACS)</td>
</tr>
<tr>
<td>n.fzg</td>
<td>Nose Gear Vertical Loads (GCS)</td>
</tr>
</tbody>
</table>
Table VII: Source Code Variable Descriptions (Continued)

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n.gx</td>
<td>Nose Gear Ground Contact FS Location</td>
</tr>
<tr>
<td>n.gy</td>
<td>Nose Gear Ground Contact FS Location</td>
</tr>
<tr>
<td>n.gz</td>
<td>Nose Gear Ground Contact FS Location</td>
</tr>
<tr>
<td>n.kt</td>
<td>Nose Gear Tire Stiffness</td>
</tr>
<tr>
<td>n.lxo</td>
<td>Nose Gear Lower Ref. Initial FS Position</td>
</tr>
<tr>
<td>n.lyo</td>
<td>Nose Gear Lower Ref. Initial BL Position</td>
</tr>
<tr>
<td>n.lzo</td>
<td>Nose Gear Lower Ref. Initial WL Position</td>
</tr>
<tr>
<td>n.oa</td>
<td>Nose Gear Oil Area</td>
</tr>
<tr>
<td>n.ov</td>
<td>Nose Gear Oil Volume</td>
</tr>
<tr>
<td>n.po</td>
<td>Nose Gear Service Pressure</td>
</tr>
<tr>
<td>n.px</td>
<td>Nose Gear Pivot Initial FS Position</td>
</tr>
<tr>
<td>n.py</td>
<td>Nose Gear Pivot Initial BL Position</td>
</tr>
<tr>
<td>n.pz</td>
<td>Nose Gear Pivot Initial WL Position</td>
</tr>
<tr>
<td>n.rt</td>
<td>Nose Gear Tire Radius</td>
</tr>
<tr>
<td>n.rto</td>
<td>Nose Gear Initial Tire Radius</td>
</tr>
<tr>
<td>n.uxo</td>
<td>Nose Gear Upper Ref. Initial FS Position</td>
</tr>
<tr>
<td>n.uyo</td>
<td>Nose Gear Upper Ref. Initial BL Position</td>
</tr>
<tr>
<td>n.uzo</td>
<td>Nose Gear Upper Ref. Initial WL Position</td>
</tr>
<tr>
<td>n.vo</td>
<td>Nose Gear Initial Air Volume</td>
</tr>
<tr>
<td>p.aa</td>
<td>Port Gear Air Area</td>
</tr>
<tr>
<td>p.ax</td>
<td>Port Gear Axle FS Position</td>
</tr>
<tr>
<td>p.axo</td>
<td>Port Gear Axle Initial FS Position</td>
</tr>
<tr>
<td>p.ay</td>
<td>Port Gear Axle FS Position</td>
</tr>
<tr>
<td>p.ayo</td>
<td>Port Gear Axle Initial BL Position</td>
</tr>
<tr>
<td>p.az</td>
<td>Port Gear Axle FS Position</td>
</tr>
<tr>
<td>p.azo</td>
<td>Port Gear Axle Initial WL Position</td>
</tr>
<tr>
<td>p.fxb</td>
<td>Port Gear Axial Loads (ACS)</td>
</tr>
<tr>
<td>p.fxg</td>
<td>Port Gear Axial Loads (GCS)</td>
</tr>
<tr>
<td>p.fyb</td>
<td>Port Gear Lateral Loads (ACS)</td>
</tr>
<tr>
<td>p.fyg</td>
<td>Port Gear Lateral Loads (GCS)</td>
</tr>
<tr>
<td>p.fzb</td>
<td>Port Gear Vertical Loads (ACS)</td>
</tr>
<tr>
<td>p.fzg</td>
<td>Port Gear Vertical Loads (GCS)</td>
</tr>
<tr>
<td>p.gx</td>
<td>Port Gear Ground Contact FS Location</td>
</tr>
<tr>
<td>p.gy</td>
<td>Port Gear Ground Contact FS Location</td>
</tr>
<tr>
<td>p.gz</td>
<td>Port Gear Ground Contact FS Location</td>
</tr>
<tr>
<td>p.kt</td>
<td>Port Gear Tire Stiffness</td>
</tr>
<tr>
<td>Variable Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td>p.lxo</td>
<td>Port Gear Lower Ref. Initial FS Position</td>
</tr>
<tr>
<td>p.lyo</td>
<td>Port Gear Lower Ref. Initial BL Position</td>
</tr>
<tr>
<td>p.lzo</td>
<td>Port Gear Lower Ref. Initial WL Position</td>
</tr>
<tr>
<td>p.oa</td>
<td>Port Gear Oil Area</td>
</tr>
<tr>
<td>p.ov</td>
<td>Port Gear Oil Volume</td>
</tr>
<tr>
<td>p.po</td>
<td>Port Gear Service Pressure</td>
</tr>
<tr>
<td>p.px</td>
<td>Port Gear Pivot Initial FS Position</td>
</tr>
<tr>
<td>p.py</td>
<td>Port Gear Pivot Initial BL Position</td>
</tr>
<tr>
<td>p.pz</td>
<td>Port Gear Pivot Initial WL Position</td>
</tr>
<tr>
<td>p.rt</td>
<td>Port Gear Tire Radius</td>
</tr>
<tr>
<td>p.rto</td>
<td>Port Gear Initial Tire Radius</td>
</tr>
<tr>
<td>p.uxo</td>
<td>Port Gear Upper Ref. Initial FS Position</td>
</tr>
<tr>
<td>p.uyo</td>
<td>Port Gear Upper Ref. Initial BL Position</td>
</tr>
<tr>
<td>p.uzo</td>
<td>Port Gear Upper Ref. Initial WL Position</td>
</tr>
<tr>
<td>p.vo</td>
<td>Port Gear Initial Air Volume</td>
</tr>
<tr>
<td>pitch</td>
<td>Aircraft Pitch Angle</td>
</tr>
<tr>
<td>rnm</td>
<td>Distance from Nose to Main Gear (contact Points)</td>
</tr>
<tr>
<td>roll</td>
<td>Aircraft Roll Angle</td>
</tr>
<tr>
<td>s.aa</td>
<td>Starboard Gear Air Area</td>
</tr>
<tr>
<td>s.ax</td>
<td>Starboard Gear Axle FS Position</td>
</tr>
<tr>
<td>s.axo</td>
<td>Starboard Gear Axle Initial FS Position</td>
</tr>
<tr>
<td>s.ay</td>
<td>Starboard Gear Axle FS Position</td>
</tr>
<tr>
<td>s.ayo</td>
<td>Starboard Gear Axle Initial BL Position</td>
</tr>
<tr>
<td>s.az</td>
<td>Starboard Gear Axle FS Position</td>
</tr>
<tr>
<td>s.azo</td>
<td>Starboard Gear Axle Initial WL Position</td>
</tr>
<tr>
<td>s.fxb</td>
<td>Starboard Gear Axial Loads (ACS)</td>
</tr>
<tr>
<td>s.fxg</td>
<td>Starboard Gear Axial Loads (GCS)</td>
</tr>
<tr>
<td>s.fyb</td>
<td>Starboard Gear Lateral Loads (ACS)</td>
</tr>
<tr>
<td>s.fyg</td>
<td>Starboard Gear Lateral Loads (GCS)</td>
</tr>
<tr>
<td>s.fzb</td>
<td>Starboard Gear Vertical Loads (ACS)</td>
</tr>
<tr>
<td>s.fzg</td>
<td>Starboard Gear Vertical Loads (GCS)</td>
</tr>
<tr>
<td>s.gx</td>
<td>Starboard Gear Ground Contact FS Location</td>
</tr>
<tr>
<td>s.gy</td>
<td>Starboard Gear Ground Contact FS Location</td>
</tr>
<tr>
<td>s.gz</td>
<td>Starboard Gear Ground Contact FS Location</td>
</tr>
<tr>
<td>s.kt</td>
<td>Starboard Gear Tire Stiffness</td>
</tr>
<tr>
<td>Variable Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>s.lxo</td>
<td>Starboard Gear Lower Ref. Initial FS Position</td>
</tr>
<tr>
<td>s.lyo</td>
<td>Starboard Gear Lower Ref. Initial BL Position</td>
</tr>
<tr>
<td>s.lzo</td>
<td>Starboard Gear Lower Ref. Initial WL Position</td>
</tr>
<tr>
<td>s.ao</td>
<td>Starboard Gear Oil Area</td>
</tr>
<tr>
<td>s.ov</td>
<td>Starboard Gear Oil Volume</td>
</tr>
<tr>
<td>s.po</td>
<td>Starboard Gear Service Pressure</td>
</tr>
<tr>
<td>s.px</td>
<td>Starboard Gear Pivot Initial FS Position</td>
</tr>
<tr>
<td>s.py</td>
<td>Starboard Gear Pivot Initial BL Position</td>
</tr>
<tr>
<td>s.pz</td>
<td>Starboard Gear Pivot Initial WL Position</td>
</tr>
<tr>
<td>s.rtx</td>
<td>Starboard Gear Pivot Initial FS Position</td>
</tr>
<tr>
<td>s.rty</td>
<td>Starboard Gear Pivot Initial BL Position</td>
</tr>
<tr>
<td>s.uzo</td>
<td>Starboard Gear Pivot Initial WL Position</td>
</tr>
<tr>
<td>s.vo</td>
<td>Starboard Gear Pivot Initial Air Volume</td>
</tr>
<tr>
<td>sinalpha</td>
<td>Sine of the Pitch Angle</td>
</tr>
<tr>
<td>t[][]</td>
<td>Transformation Matrix</td>
</tr>
<tr>
<td>w.deltaCG</td>
<td>Delta CG for Graphics</td>
</tr>
<tr>
<td>w.deltaWT</td>
<td>Delta WT for Graphics</td>
</tr>
<tr>
<td>w.fcG#</td>
<td>Fuel Burn CG Points</td>
</tr>
<tr>
<td>w.fWT#</td>
<td>Fuel Burn WT Points</td>
</tr>
<tr>
<td>w.mac</td>
<td>Mean Aerodynamic Chord</td>
</tr>
<tr>
<td>w.macL</td>
<td>Leading Edge of the MAC</td>
</tr>
<tr>
<td>w.offsetCG</td>
<td>Offset CG for Graphics</td>
</tr>
<tr>
<td>w.offsetWT</td>
<td>Offset WT for Graphics</td>
</tr>
<tr>
<td>w.zfCG#</td>
<td>Zero Fuel Envelope CGs</td>
</tr>
<tr>
<td>w.zfWT#</td>
<td>Zero Fuel Envelope WTs</td>
</tr>
<tr>
<td>xt</td>
<td>Aircraft Weight</td>
</tr>
<tr>
<td>yaw</td>
<td>Aircraft Yaw Angle</td>
</tr>
</tbody>
</table>
APPENDIX C
SOURCE CODE

The listings of each source code file of the GASP Utility is provided below. Table VIII provides a list of the filenames and the order in which they are listed.

Table VIII: Source Code Listing

<table>
<thead>
<tr>
<th>Class</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ACDataServer.java</td>
</tr>
<tr>
<td>2</td>
<td>ACDataServerImpl.java</td>
</tr>
<tr>
<td>3</td>
<td>StaticGear.java</td>
</tr>
<tr>
<td>4</td>
<td>ACdata.java</td>
</tr>
<tr>
<td>5</td>
<td>Gear.java</td>
</tr>
<tr>
<td>6</td>
<td>StrokeDist.java</td>
</tr>
<tr>
<td>7</td>
<td>WtCGcanvas.java</td>
</tr>
</tbody>
</table>

ACDataServer.java

```java
import java.rmi.*;
import java.util.*;

public interface ACDataServer extends Remote
{
    public HashMap getAircraftInfo( String acType ) throws RemoteException;
}
```
ACDataServerImpl.java

import java.rmi.*;
import java.rmi.server.*;
import java.util.*;
import java.io.*;
import java.net.*;

public class ACDataServerImpl extends UnicastRemoteObject
        implements ACDataServer
{
private HashMap gx = new HashMap();
public ACDataServerImpl() throws RemoteException
{
    super();
}
// get one set of aircraft information
private HashMap updateAircraftData( String aircraft )
        throws RemoteException
{
    HashMap ac = new HashMap();
    try
    {
        System.err.println("Retrieving " + aircraft + " Aircraft information...");
        BufferedReader in = new BufferedReader(new InputStreamReader(
                new FileInputStream(aircraft + "data.txt")));

        // skip header above aircraft information
        String inputLine = "";
        do { inputLine = in.readLine(); }
            while ( inputLine.startsWith("//") );
        //System.err.println( inputLine );
        StringTokenizer tokens = new StringTokenizer(
                inputLine);
        //System.err.println( "Finished Processing Data." );
        in.close(); // close connection to data file
        // code to check hashmap contents
        // System.err.println( ac.keySet() );

        while ( !inputLine.equals( "end" ) )
        {
            // add to HashMap for the selected aircraft
            ac.put( tokens.nextToken(), new Double( Double.parseDouble( tokens.nextToken() ) ) );
            // check for and skip comments comments
            do { inputLine = in.readLine(); }
                while ( inputLine.startsWith("//") );
            tokens = new StringTokenizer( inputLine );
            //System.err.println( inputLine );
        }
    }
}

// code to check hashmap contents
System.err.println( "Finished Processing Data." );
in.close(); // close connection to data file
// System.err.println( ac.keySet() );
// System.err.println( ac.values() );
// System.err.println( ac.entrySet() );
}
catch( java.net.ConnectException ce )
{
    System.err.println( "Connection failed." );
    System.exit( 1 );
}
catch( Exception e )
{
    e.printStackTrace();
    System.exit( 1 );
}
return ac;

// implementation for ACDataServer interface method
public HashMap getAircraftInfo( String acType ) throws RemoteException
{
    System.err.println( "Aircraft Data Sent = " + acType );
    // the check for valid AC = Client can only sent a valid filename (hardwired in pulldown)
    gx = updateAircraftData( acType );
    return gx;
}

class ACDataServerServerImpl
{
    public static void main( String args[] ) throws Exception
    {
        System.err.println( "Initializing server: please wait." );
        // create server object
        ACDataServerServerImpl aircraft = new ACDataServerServerImpl();
        // bind ACDataServerServerImpl object to the rmiregistry
        String serverObjectName = "//localhost/DataServer";
        // String serverObjectName = 
        // "//LAPTOP.sav.gulfaero.com/DataServer";
        // String serverObjectName = 
        // "//steed-100.catia.gulfaero.com/DataServer";
        Naming.rebind( serverObjectName, aircraft );
        System.err.println( "The ACData Server is up and running." );
    }
}
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.text.DecimalFormat;
import java.rmi.*;
import java.io.*;
import java.util.*;
import java.lang.*;
import java.lang.reflect.*;

public class StaticGear extends JApplet implements ActionListener {

    // Create Major Objects
    ACdata a = new ACdata(this);
    Gear n = new Gear();
    StrokeDist d = new StrokeDist();

    Gear p = new Gear();
    WtCGcanvas w = new WtCGcanvas(a, this);
    Gear s = new Gear();

    HashMap ac = new HashMap();
    boolean firstAircraft = true;
    public double cg = 0.0;
    public double wt = 0.0;
    public int ifu = 0;
    public double[] fuelburnCG = { 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0 };
    public double[] fuelburnWT = { 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0 };
    public double roll = 0.0;
    public double pitch = 0.0;
    public double yaw = 0.0;
    public double grd[] = { 0.0, 0.0, 0.0, 0.0 };
    public double t[][] = {{ 0.0, 0.0, 0.0, 0.0 },
                           { 0.0, 0.0, 0.0, 0.0 },
                           { 0.0, 0.0, 0.0, 0.0 }};

    DecimalFormat twodig = new DecimalFormat( "0.00" );
    DecimalFormat fourdig = new DecimalFormat( "0.0000" );

    public void init()
    {
        Container contentPane = getContentPane();
        contentPane.setLayout (new GridLayout(2, 3));
// Add Borders
p.setBorder(BorderFactory.createLineBorder(Color.black));
n.setBorder(BorderFactory.createLineBorder(Color.black));
s.setBorder(BorderFactory.createLineBorder(Color.black));
// Place Major Objects on contentpane
contentPane.add(a);
contentPane.add(n);
contentPane.add(d);
contentPane.add(p);
contentPane.add(w);
contentPane.add(s);

// Add listener for A/C selection
a.tmodel.addActionListener(new ActionListener()
{
   public void actionPerformed(ActionEvent e)
   {
      Object acTypeo = a.tmodel.getSelectedItem();
      String acType = acTypeo.toString();

      if( acType.equals("Select A/C") )
         JOptionPane.showMessageDialog(null,"Select an Aircraft Type from the Drop List");
      else
      {
         n.lgear.setText(acType + " NOSE");
         p.lgear.setText(acType + " PORT");
         s.lgear.setText(acType + " STAR");

         // String ip = "steed-100.catia.gulfaero.com";
         // String ip = "LAPTOP.sav.gulfaero.com";
         String ip = "localhost";

         getRemoteData( acType , ip );
         initresults();
         calResults();
      }
   }
});
//initresults();

public void setwtcg(double wt, double cg)
{
   this.wt = wt;
   this.cg = cg;
   a.zfgw = this.wt;
   a.zfcg = this.cg;
}
public void setfuel(int ifu)
{
    this.ifu = ifu;

    // Calculate fuel weight
    a.fuelw = this.ifu / 100.0 * (w.fWT9 - 1.0);   // fuel slider sends % max fuel
    // Calculate Fuel CG as a function of the fuel weight
    int i = 0;
    while (fuelburnWT[i] <= a.fuelw) i++;
    a.fuelcg = fuelburnCG[i-1] + ((fuelburnCG[i] - fuelburnCG[i-1]) * (a.fuelw - fuelburnWT[i-1])) / (fuelburnWT[i] - fuelburnWT[i-1]);
    //System.err.println( a.fuelcg + "\t" + a.fuelw );
}

public void calResults()
{
    // calculate gear positions and loads
    double alphaax;
    double alphat;
    double rnm;
    double sinalpha;
    double cosalpha;

    n.rt = n.rto;
    p.rt = p.rto;
    s.rt = s.rto;

    // calculate AC gross weight
    a.gw = a.zfgw + a.fuelw;
    // calculate AC XCG (in)
    double zfci = w.macLE + w.mac / 100.0 * a.zfcg;
    a.cgi = ( (zfci * a.zfgw) + (a.fuelcg * a.fuelw) ) / a.gw;
    // set AC ZCG to constant (60 in)
    a.cgz = ( (a.zcgz * a.zfgw) + (a.zcgf * a.fuelw) ) / a.gw;

    // calculate AC XCG in %MAC
    a.cgp = 100.0 * (a.cgi - w.macLE) / w.mac;

    // Display Weight CG data
    a.tzfgw.setText( twodig.format(a.zfgw) );
    a.tzfcg.setText( twodig.format(a.zfcg) );
    a.tfuelw.setText( twodig.format(a.fuelw) );
    a.tfuelcg.setText( twodig.format(a.fuelcg) );
    a.tgw.setText( twodig.format(a.gw) );
for ( int i=0; i<6; i++ )
{
    // compute nose gear kinematics
    d.nstr = Math.max(0.0,d.nstr);
    n.ax = n.axo + d.nstr * Math.sin(n.theta);
    n.az = n.azo + d.nstr * Math.cos(n.theta);

    // compute main gear kinematics
    d.sstr = Math.max(0.0,d.sstr);
    s.th3 = Math.acos( (s.rpl * s.rpl+s.rpu * s.rpu -
                     (s.rul - d.sstr) * (s.rul - d.sstr))/((2.0 * s.rpl * s.rpu) ));
    s.thax = s.theta+s.thetaUPL - s.th3;
    s.ax = s.px+s.rpa * Math.cos(s.thax);
    s.az = s.pz+s.rpa * Math.sin(s.thax);

    // compute symmetric aircraft kinematics
    rnm = Math.sqrt( (s.ax - n.ax) * (s.ax - n.ax) *
                     (s.az - n.az) * (s.az - n.az) );

    //System.out.println("nose to main =" + rnm);
    alphaax = Math.atan( (s.az - n.az)/(s.ax - n.ax) );
    alphat = Math.asin( (n.rt - s.rt)/rnm );
    pitch = alphaax+alphat;
    sinalpha = Math.sin(pitch);
    cosalpha = Math.cos(pitch);

    n.gx = n.ax + n.rt * sinalpha;
    n.gz = n.az - n.rt * cosalpha;
    s.gx = s.ax + s.rt * sinalpha;
    s.gz = s.az - s.rt * cosalpha;

    // compute aircraft equilibrium (symmetric)
    s.fzg = a.gw / 2. * a.nz * ( ((a.cgi - n.gx) *
                             cosalpha +a.cgz - n.gz) * sinalpha) / (rnm *
                             Math.cos(alphat) + a.mu * ((a.cgz - n.gz) *
                             cosalpha - (a.cgi - n.gx) * sinalpha)) );
    s.fxg = a.mu * s.fzg;
    n.fzg = a.gw * a.nz - 2. * s.fzg;

    // compute tire radii
    n.rt = n.rto - n.fzg / n.kt;
    s.rt = s.rto - s.fzg / s.kt;

    // NLG oleo calculations
    n.fo = n.fzg * Math.cos(n.theta + pitch);
n.pp = Math.max(n.fo / n.aa, n.po);
d.nstr = n.vo / n.aa * (1.0 - (n.po + a.ampress) / (n.pp + a.ampress));

// MLG oleo calculations
s.fo = s.fzg * ((s.rpa * (Math.cos(s.thax - pitch) -
  a.mu * Math.sin(s.thax - pitch)) +
  a.mu * s.rt) * (s.rul - d.sstr)) /
  (s.rpl * s.rpu * Math.sin(s.th3));
s.pp = Math.max(s.fo / s.aa, s.po);
d.sstr = s.vo / s.aa * (1.0 - (s.po + a.ampress) / (s.pp + a.ampress));
}

// Set Transformation matrix
setTrans();

// Ground Contact Location
n.gx = n.ax + n.rt * Math.sin(pitch);
n.gy = n.ayo;
n.gz = n.az - n.rt * Math.cos(pitch);

s.gx = s.ax + s.rt * Math.sin(pitch);
s.gy = s.ayo;
s.gz = s.az - s.rt * Math.cos(pitch);

p.gx = s.gx;
p.gy = -s.gy;
p.gz = s.gz;

// Axle Location
n.ay = n.ayo; // n.ax and n.az calculated in main loop
s.ay = s.ayo; // s.ax and s.az calculated in main loop

p.ax = s.ax;
p.ay = -s.ay;
p.az = s.az;

// Ground Axis Gear Loads
n.fyg = 0.0; // for symmetric solution, n.fxg and
  n.fzg calculated in main loop
s.fyg = 0.0; // for symmetric solution, s.fxg and
  s.fzg calculated in main loop

p.fxg = s.fxg;
p.fyg = s.fyg;
p.fzg = s.fzg;

// Body Axis Gear Loads
doTrans(0.0, 0.0, 0.0, n.fxg, n.fyg, n.fzg);
n.fxb = grd[0];
n.fyb = grd[1];
n.fzb = grd[2];
doTrans(0.0, 0.0, 0.0, s.fxg, s.fyg, s.fzg);
s.fxg = grd[0];
s.fyg = grd[1];
s.fzg = grd[2];

doTrans(0.0, 0.0, 0.0, p.fxg, p.fyg, p.fzg);
p.fxg = grd[0];
p.fyg = grd[1];
p.fzg = grd[2];

// Nx
a.nx = 2. * s.fxg/a.gw;  // check changed s.Fd to s.fxg

//compute ground height
// h=(ix-s.gx)*sinalpha+(iz-s.gz)*cosalpha;
doTrans(n.gx, n.gy, n.gz, d.wltx, d.wlty, d.wltz);
d.twltd.setText (twodig.format(grd[2]));

doTrans(n.gx, n.gy, n.gz, d.vttx, d.vtty, d.vttz);
d.tvtttd.setText (twodig.format(grd[2]));

doTrans(n.gx, n.gy, n.gz, d.rdx, d.rdy, d.rdz);
d.trdd.setText (twodig.format(grd[2]));

doTrans(n.gx, n.gy, n.gz, d.selx, d.sely, d.selz);
d.tseld.setText (twodig.format(grd[2]));

// calculate deck and roll angles
a.decka = pitch * 180.0 / 3.14159;
a.rolla = 0.0;  // for Symmetric Solution

// calculate % stroke
d.pstr = d.sstr;
d.nstrp = d.nstr / a.mngs * 100.;
d.sstrp = d.sstr / a.mngs * 100.;
d.pstrp = d.pstr / a.mmgs * 100.;

// output results
// angles
a.tdecka.setText(twodig.format(a.decka));
a.trolla.setText(twodig.format(a.rolla));

// strokes
d.tpstr.setText (twodig.format(d.pstr));
d.tpstrp.setText(twodig.format(d.pstrp));
d.tnstr.setText (twodig.format(d.nstr));
d.tnstrp.setText(twodig.format(d.nstrp));
d.tsstr.setText (twodig.format(d.sstr));
d.tsstrp.setText(twodig.format(d.sstrp));

// Nose
n.tfxg.setText(twodig.format(n.fxg));
n.tfyg.setText(twodig.format(n.fyg));
n.tfzg.setText(twodig.format(n.fzg));

n.tfxb.setText(twodig.format(n.fxb));
n.tfyb.setText(twodig.format(n.fyb));
n.tfzb.setText(twodig.format(n.fzb));

n.tax.setText(twodig.format(n.ax));
n.tay.setText(twodig.format(n.ay));
n.taz.setText(twodig.format(n.az));

n.tgx.setText(twodig.format(n.gx));
n.tgy.setText(twodig.format(n.gy));
n.tgz.setText(twodig.format(n.gz));

// Port
p.tfxg.setText(twodig.format(p.fxg));
p.tfyg.setText(twodig.format(p.fyg));
p.tfzg.setText(twodig.format(p.fzg));

p.tfxb.setText(twodig.format(p.fxb));
p.tfyb.setText(twodig.format(p.fyb));
p.tfzb.setText(twodig.format(p.fzb));

p.tax.setText(twodig.format(p.ax));
p.tay.setText(twodig.format(p.ay));
p.taz.setText(twodig.format(p.az));

p.tgx.setText(twodig.format(p.gx));
p.tgy.setText(twodig.format(p.gy));
p.tgz.setText(twodig.format(p.gz));

// Starboard
s.tfxg.setText(twodig.format(s.fxg));
s.tfyg.setText(twodig.format(s.fyg));
s.tfzg.setText(twodig.format(s.fzg));

s.tfxb.setText(twodig.format(s.fxb));
s.tfyb.setText(twodig.format(s.fyb));
s.tfzb.setText(twodig.format(s.fzb));

s.tax.setText(twodig.format(s.ax));
s.tay.setText(twodig.format(s.ay));
s.taz.setText(twodig.format(s.az));

s.tgx.setText(twodig.format(s.gx));
s.tgy.setText(twodig.format(s.gy));
s.tgz.setText(twodig.format(s.gz));

// set stroke sliders
d.spstrp.setValue( (int) Math.round(d.pstrp) );
public void setTrans()
{
    t[0][0] = Math.cos(-pitch)*Math.cos(yaw);
    t[0][1] = Math.cos(-pitch)*Math.sin(yaw);
    t[0][2] = -Math.sin(-pitch);
    
    t[1][0] = Math.sin(roll)*Math.sin(-pitch)*Math.cos(yaw)
            - Math.cos(roll)*Math.sin(yaw);
    t[1][1] = Math.sin(roll)*Math.sin(-pitch)*Math.sin(yaw)
            + Math.cos(roll)*Math.cos(yaw);
    t[1][2] = Math.sin(roll)*Math.cos(-pitch);
    
    t[2][0] = Math.cos(roll)*Math.sin(-pitch)*Math.cos(yaw)
            + Math.sin(roll)*Math.sin(yaw);
            - Math.sin(roll)*Math.cos(yaw);
    t[2][2] = Math.cos(roll)*Math.cos(-pitch);
}

public void doTrans( double xl, double yl, double zl,
                      double x2, double y2, double z2 )
{
    double grd0[] = { xl,yl,zl }; 
    double acs[] = { x2,y2,z2 }; 

    for (int j = 0; j <= 2; j++)
    {
        grd[j] = 0.0;
        acs[j] = acs[j] - grd0[j]; 
        for (int k = 0; k <= 2; k++)
        {
            grd[j] = grd[j] + acs[k] * t[j][k];
        }
        //grd[j] = grd[j] - grd0[j];
    }
}

private void getRemoteData( String acType, String ip )
{
    try
    {
        // name of remote server object bound to rmi registry
        String serverObjectName = "//" + ip + "/DataServer";

        // lookup ACDataServerImpl remote object in rmiregistry
        ACDataServer mydata = ( ACDataServer ) Naming.lookup(
                              serverObjectName );
    }
Get remote ACInfo object from Server
ac = mydata.getAircraftInfo( acType );

Check HashMap Contents
System.err.println( ac.keySet() );
System.err.println( ac.values() );
System.err.println( ac.entrySet() );

setAircraftValues( n, p, s, a, d, w, ac );

} catch ( java.rmi.ConnectException ce )
{
    System.err.println( "Connection to server failed. " +
                       "Server may be temporarily unavailable." );
}

} catch ( Exception e )
{
    e.printStackTrace();
    System.exit(0);
}

private void setAircraftValues(
    Gear cn, Gear cp, Gear cs,
    ACdata ca, StrokeDist cd, WtCGcanvas cw,
    HashMap map ) throws Exception
{
    // Get Classes
    Class ccg = cn.getClass();
    Class cca = ca.getClass();
    Class ccd = cd.getClass();
    Class ccw = cw.getClass();

    // get the set of HashMap keys
    Set s = map.keySet();
    // initiate iterator using key set
    Iterator i = s.iterator();
    // loop on key set
    while( i.hasNext() )
    {
        // get the next key as a string
        String fieldName = (String)i.next();
        //System.err.println( fieldName );
        // get the value associated with the key
        double value = ((Double)map.get( fieldName )).doubleValue();

        // work with what instance
        if( fieldName.startsWith( "n." ) ) // nose gear = n
        {
            // field (variable) with same name as key (less the
2 char instance prefix)
Field f = ccg.getField( fieldName.substring( 2 ) );
// set the variable to value associated with the
// field (JTextField) with same name as key with t
// prefix less the 2 char instance prefix
f = ccg.getField( "t" + fieldName.substring( 2 ) );
// get the JTextField
JTextField jtf = (JTextField)f.get( cn );
// set the JTextField to the Variable value
jtf.setText( twodig.format( value ) );
}

else if( fieldName.startsWith( "p." ) ) // port gear = p
{
    Field f = ccg.getField( fieldName.substring( 2 ) );
    f.setDouble( cp, value );
    f = ccg.getField( "t" + fieldName.substring( 2 ) );
    JTextField jtf = (JTextField)f.get( cp );
    jtf.setText( twodig.format( value ) );
    jtf.setHorizontalAlignment( javax.swing.SwingConstants.RIGHT);
    if( firstAircraft ) jtf.addActionListener( this );
}

else if( fieldName.startsWith( "s." ) ) // starboard
gear = s
{
    Field f = ccg.getField( fieldName.substring( 2 ) );
    f.setDouble( cs, value );
    f = ccg.getField( "t" + fieldName.substring( 2 ) );
    JTextField jtf = (JTextField)f.get( cs );
    jtf.setText( twodig.format( value ) );
    jtf.setHorizontalAlignment( javax.swing.SwingConstants.RIGHT);
    if( firstAircraft ) jtf.addActionListener( this );
}

else if( fieldName.startsWith( "a." ) ) // ACdata
{
    Field f = cca.getField( fieldName.substring( 2 ) );
    f.setDouble( ca, value );
    f = cca.getField( "t" + fieldName.substring( 2 ) );
    JTextField jtf = (JTextField)f.get( ca );
    jtf.setText( twodig.format( value ) );
    jtf.setHorizontalAlignment( javax.swing.SwingConstants.RIGHT);
if( firstAircraft ) jtf.addActionListener( this );
}
else if(fieldName.startsWith( "d." ) ) // StrokeDist
{
    Field f = ccd.getField( fieldName.substring( 2 ) );
    f.setDouble( cd, value );
    f = ccd.getField( "t" + fieldName.substring( 2 ) );
    JTextField jtf = (JTextField)f.get( cd );
    jtf.setText( twodig.format( value ) );
    jtf.setHorizontalAlignment( javax.swing.SwingConstants.RIGHT);
    if( firstAircraft ) jtf.addActionListener( this );
}
else if(fieldName.startsWith( "w." ) ) // WtCGcanvas
{
    Field f = ccw.getField( fieldName.substring( 2 ) );
    f.setDouble( cw, value );
}
else  // anything else
{
    JOptionPane.showMessageDialog( null,
    "Key: " + fieldName + " with value: " + value + "
has no Variable Match\nClient will now Stop" );
    System.exit( 1 );
}
firstAircraft = false;
}

private void updateVariables(Gear cn, Gear cp, Gear cs, ACdata ca, StrokeDist cd, WtCGcanvas cw, HashMap map ) throws Exception
{
    // Get Classes
    Class ccg = cn.getClass();
    Class cca = ca.getClass();
    Class ccd = cd.getClass();
    Class ccw = cw.getClass();

    // get the set of HashMap keys
    Set s = map.keySet();
    // initiate iterator using key set
    Iterator i = s.iterator();
    // loop on key set
    while( i.hasNext() )
    {
        // get the next key as a string
        String fieldName = (String)i.next();
        }
// work with what instance
if (fieldName.startsWith( "n."))  // nose gear = n
{
    Field f1 = ccg.getField( fieldName.substring( 2 ) );
    Field f2 = ccg.getField( "t" + fieldName.substring( 2 ) );
    JTextField jtf = (JTextField)f2.get( cn );
    double value = Double.parseDouble( jtf.getText() );
    f1.setDouble( cn, value );
} else if (fieldName.startsWith( "p."))  // port gear = p
{
    Field f1 = ccg.getField( fieldName.substring( 2 ) );
    Field f2 = ccg.getField( "t" + fieldName.substring( 2 ) );
    JTextField jtf = (JTextField)f2.get( cp );
    double value = Double.parseDouble( jtf.getText() );
    f1.setDouble( cp, value );
} else if (fieldName.startsWith( "s."))  // starboard gear = s
{
    Field f1 = ccg.getField( fieldName.substring( 2 ) );
    Field f2 = ccg.getField( "t" + fieldName.substring( 2 ) );
    JTextField jtf = (JTextField)f2.get( cs );
    double value = Double.parseDouble( jtf.getText() );
    f1.setDouble( cs, value );
} else if (fieldName.startsWith( "a."))  // ACdata
{
    Field f1 = cca.getField( fieldName.substring( 2 ) );
    Field f2 = cca.getField( "t" + fieldName.substring( 2 ) );
    JTextField jtf = (JTextField)f2.get( ca );
    double value = Double.parseDouble( jtf.getText() );
    f1.setDouble( ca, value );
} else if (fieldName.startsWith( "d."))  // StrokeDist
{
    Field f1 = ccd.getField( fieldName.substring( 2 ) );
    Field f2 = ccd.getField( "t" + fieldName.substring( 2 ) );
    JTextField jtf = (JTextField)f2.get( cd );
    double value = Double.parseDouble( jtf.getText() );
    f1.setDouble( cd, value );
}

// If any JTextField is modified, all user defined
variables are updated
public void actionPerformed ( ActionEvent ev )
{
    try
    {
        // add check for valid entry ( a number )
        ******************************
        upDateVariables ( n, p, s, a, d, w, ac );
        w.setZFGWpt ( a.zfcg, a.zfgw );
        calResults();
    }
    catch ( Exception e )
    {
        e.printStackTrace();
        System.exit (0);
    }
}

public void initresults ()
{
    // Load Zero Fuel Envelope Polygon
    w.setCanvas ( w.deltaCG, w.deltaWT, w.offsetCG,
                 w.offsetWT );
    Polygon acZFE = new Polygon();
    acZFE.addPoint ( (int) w.ix ( w.zfCG1 ), (int) w.iy ( w.zfWT1 ));
    acZFE.addPoint ( (int) w.ix ( w.zfCG2 ), (int) w.iy ( w.zfWT2 ));
    acZFE.addPoint ( (int) w.ix ( w.zfCG3 ), (int) w.iy ( w.zfWT3 ));
    acZFE.addPoint ( (int) w.ix ( w.zfCG4 ), (int) w.iy ( w.zfWT4 ));
    acZFE.addPoint ( (int) w.ix ( w.zfCG5 ), (int) w.iy ( w.zfWT5 ));
    acZFE.addPoint ( (int) w.ix ( w.zfCG6 ), (int) w.iy ( w.zfWT6 ));
    acZFE.addPoint ( (int) w.ix ( w.zfCG7 ), (int) w.iy ( w.zfWT7 ));
    acZFE.addPoint ( (int) w.ix ( w.zfCG8 ), (int) w.iy ( w.zfWT8 ));
    w.setZFGWE ( acZFE );
    w.setZFGWpt ( a.zfcg, a.zfgw );
    // Load Fuel Burn Curve
    fuelburnCG[ 0 ] = w.fCG1;
    fuelburnCG[ 1 ] = w.fCG2;
    fuelburnCG[ 2 ] = w.fCG3;
    fuelburnCG[ 3 ] = w.fCG4;
    fuelburnCG[ 4 ] = w.fCG5;
    fuelburnCG[ 5 ] = w.fCG6;
fuelburnCG[ 6 ] = w.fCG7;
fuelburnCG[ 7 ] = w.fCG8;
fuelburnCG[ 8 ] = w.fCG9;

fuelburnWT[ 0 ] = w.fWT1;
fuelburnWT[ 1 ] = w.fWT2;
fuelburnWT[ 2 ] = w.fWT3;
fuelburnWT[ 3 ] = w.fWT4;
fuelburnWT[ 4 ] = w.fWT5;
fuelburnWT[ 5 ] = w.fWT6;
fuelburnWT[ 6 ] = w.fWT7;
fuelburnWT[ 7 ] = w.fWT8;
fuelburnWT[ 8 ] = w.fWT9;

// set initial slider values
// strokes
//d.spstrp.setValue ( 100 );
//d.snstrp.setValue ( 100 );
//d.ssstrp.setValue ( 100 );
// fuel
a.sfuel.setValue ( 0 );
a.tfuelcg.setText( "0.00" );

// Calculate A/C dependent Geometry
// nose
n.theta = Math.atan( (n.uxo - n.Ixo)/(n.uzo - n.Izo) );
// port
       * (p.py - p.ayo)+(p.pz - p.azo) * (p.pz - p.azo) );
       * (p.py - p.lyo)+(p.pz - p.lzo) * (p.pz - p.lzo) );
       * (p.py - p.uyo)+(p.pz - p.azo) * (p.pz - p.azo) );
       * (p.lyo - p.uyo)+(p.lzo - p.azo) * (p.lzo - p.azo) );
p.theta = Math.atan( (p.azo - p.pz)/(p.axo - p.px) );
p.thetaUPL = Math.acos( ( (p.uxo - p.pz) * (p.lxo - p.pz)
       * (p.lzo - p.pz) ) / (p.rpl * p.rpu) );
// star
s.rpa = Math.sqrt( (s.px - s.axo) * (s.px - s.axo)+(s.py - s.ayo)
* (s.py - s.ayo)+(s.pz - s.azo) * (s.pz - s.azo) );
s.rpl = Math.sqrt( (s.px - s.lxo) * (s.px - s.lxo)+(s.py - s.lyo)
* (s.py - s.lyo)+(s.pz - s.lzo) * (s.pz - s.lzo) );
s.rpu = Math.sqrt( (s.px - s.uxo) * (s.px - s.uxo)+(s.py - s.uyo)
* (s.py - s.uyo)+(s.pz - s.uzo) * (s.pz - s.uzo) );
s.rul = Math.sqrt( (s.lxo - s.uxo) * (s.lxo - s.uxo)+(s.lyo - s.uyo)
* (s.lyo - s.uyo)+(s.lzo - s.uzo) * (s.lzo - s.uzo) );
s.theta = Math.atan( (s.azo - s.pz)/(s.axo - s.px) );
s.thetaUPL = Math.acos( ( (s.uxo - s.px) * (s.lxo - s.pz)+(s.uzo - s.pz)
* (s.lzo - s.pz) ) / (s.rpl * s.rpu) );
ACdata.java

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import java.text.DecimalFormat;
import java.rmi.*;
import java.io.*;
import java.util.*;
import java.lang.*;
import java.lang.reflect.*;

public class StaticGear extends JApplet implements ActionListener {

    // Create Major Objects
    ACdata a = new ACdata(this);
    Gear n = new Gear();
    StrokeDist d = new StrokeDist();

    Gear p = new Gear();
    WtCGcanvas w = new WtCGcanvas(a, this);
    Gear s = new Gear();

    HashMap ac = new HashMap();
    boolean firstAircraft = true;
    public double cg = 0.0;
    public double wt = 0.0;
    public int ifu = 0;
    public double[] fuelburnCG = { 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
                                  0.0, 0.0, 0.0, 0.0, 0.0 };
    public double[] fuelburnWT = { 0.0, 0.0, 0.0, 0.0, 0.0,
                                  0.0, 0.0, 0.0, 0.0, 0.0 };
    public double roll = 0.0;
    public double pitch = 0.0;
    public double yaw = 0.0;
    public double[] grd[] = { 0.0, 0.0, 0.0, 0.0,
                              0.0, 0.0, 0.0, 0.0, 0.0 };
    public double[][] t[][] = {{ 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
                              0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0 };
    public double twodig = new DecimalFormat( "0.00" );
    public double fourdig = new DecimalFormat( "0.0000" );

    public void init() {
        Container contentPane = getContentPane();
        contentPane.setLayout (new GridLayout(2, 3));
// Add Borders

p.setBorder(BorderFactory.createLineBorder(Color.black));

n.setBorder(BorderFactory.createLineBorder(Color.black));

s.setBorder(BorderFactory.createLineBorder(Color.black));

// Place Major Objects on contentpane
contentPane.add(a);
contentPane.add(n);
contentPane.add(d);

contentPane.add(p);
contentPane.add(w);
contentPane.add(s);

// Add listener for A/C selection
a.tmodel.addActionListener(new ActionListener()
{
  public void actionPerformed(ActionEvent e)
  {

    Object acTypeo = a.tmodel.getSelectedItem();
    String acType = acTypeo.toString();

    if( acType.equals ( "Select A/C" ) )
    JOptionPane.showMessageDialog( null,"Select an Aircraft Type from the Drop List");
    else
    {
      n.lgear.setText(acType + " NOSE");
      p.lgear.setText(acType + " PORT");
      s.lgear.setText(acType + " STAR");

      // String ip = "steed-100.catia.gulfaero.com";
      // String ip = "LAPTOP.sav.gulfaero.com";
      String ip = "localhost";
      getRemoteData( acType , ip );

      initresults();
      calResults();
    }
  }
});
public void setwtcg(double wt, double cg) {
    this.wt = wt;
    this.cg = cg;
    a.zfgw = this.wt;
    a.zfcg = this.cg;
}

public void setfuel(int ifu) {
    this.ifu = ifu;
    // Calculate fuel weight
    a.fuelw = this.ifu / 100.0 * ( w.fWT9 - 1.0 );  // fuel slider sends % max fuel
    // Calculate Fuel CG as a function of the fuel weight
    int i = 0;
    while ( fuelburnWT[ i ] <= a.fuelw ) i++;
    a.fuelcg = fuelburnCG[i-1] + (((fuelburnCG[i] - fuelburnCG[i-1]) * (a.fuelw - fuelburnWT[i-1])) / (fuelburnWT[i] - fuelburnWT[i-1]));
    //System.err.println( a.fuelcg + "\t" + a.fuelw );
}

public void calResults() {
    // calculate gear positions and loads
    double alphaax;
    double alphat;
    double rnm;
    double sinalpha;
    double cosalpha;
    n.rt = n.rto ;
    p.rt = p.rto ;
    s.rt = s.rto ;
    // calculate AC gross weight
    a.gw = a.zfgw + a.fuelw;
    // calculate AC XCG (in)
    double zfcgi = w.macLE + w.mac / 100.0 * a.zfcg;
    a.cgi = ( ( zfcgi * a.zfgw ) + ( a.fuelcg * a.fuelw ) )
}
/ a.gw;
// set AC ZCG to constant (60 in)
a.cgz = ( (a.zcgz * a.zfgw) + (a.zcgf * a.fuelw) ) / a.gw;

// calculate AC XCG in %MAC
a.cgp = 100.0 * (a.cgi - w.macLE) / w.mac;

// Display Weight CG data
a.tzfgw.setText (twodig.format(a.zfgw));
a.tzfcg.setText (twodig.format(a.zfcg));
a.tfuelw.setText (twodig.format(a.fuelw));
a.tfuelcg.setText (twodig.format(a.fuelcg));
a.tgw.setText (twodig.format(a.gw));
a.tcgp.setText (twodig.format(a.cgp));
a.tcgi.setText (twodig.format(a.cgi));

for ( int i=0; i<6; i++ )
{
    // compute nose gear kinematics
d.nstr = Math.max(0.0,d.nstr);
n.ax = n.axo + d.nstr * Math.sin(n.theta);
n.az = n.azo + d.nstr * Math.cos(n.theta);

    // compute main gear kinematics
d.sstr = Math.max(0.0,d.sstr);
s.th3 = Math.acos( (s.rpl * s.rpl+s.rpu * s.rpu -
                         (s.rul - d.sstr) * (s.rul - d.sstr))/(2.0 * s.rpl * s.rpu) );
s.thax = s.theta+s.thetaUPL - s.th3;
s.ax = s.px+s.rpa * Math.cos(s.thax);
s.az = s.pz+s.rpa * Math.sin(s.thax);

    // compute symmetric aircraft kinematics
rnnm = Math.sqrt( (s.ax - n.ax) * (s.ax - n.ax) +
                  (s.az - n.az) * (s.az - n.az) );

    //System.out.println("nose to main =" + rnm);

    alphaax = Math.atan( (s.az - n.az)/(s.ax - n.ax) );
alphat = Math.asin( (n.rt - s.rt)/rnnm );
pitch = alphaax+alphat;
sinalpha = Math.sin(pitch);
cosalpha = Math.cos(pitch);

    n.gx = n.ax + n.rt * sinalpha;
    n.gz = n.az - n.rt * cosalpha;
    s gx = s.ax + s.rt * sinalpha;
    s.gz = s.az - s.rt * cosalpha;
// compute aircraft equilibrium (symmetric)
s.fzg = a.gw / 2. * a.nz * ( ((a.cgi - n.gx) * cosalpha + (a.cgz - n.gz) * sinalpha) / (rnm * Math.cos(alphat) + a.mu * ((a.cgz - n.gz) * cosalpha - (a.cgi - n.gx) * sinalpha) ));
s.fxg = a.mu * s.fzg;
n.fzg = a.gw * a.nz - 2. * s.fzg;

// compute tire radii
n.rt = n.rto - n.fzg / n.kt;
s.rt = s.rto - s.fzg / s.kt;

// NLG oleo calculations
n.fo = n.fzg * Math.cos(n.theta + pitch);
n.pp = Math.max(n.fo / n.aa, n.po);
d.nstr = n.vo/n.aa * (1.0 - (n.po + a.ampress) / (n.pp + a.ampress));

// MLG oleo calculations
s.fo = s.fzg * ( (s.rpa * (Math.cos(s.thax - pitch) - a.mu * Math.sin(s.thax - pitch)) + a.mu * s.rt) * (s.rul - d.sstr)) / (s.rpl * s.rpu * Math.sin(s.th3));
s.pp = Math.max(s.fo / s.aa, s.po);
d.sstr = s.vo/s.aa * (1.0 - (s.po + a.ampress) / (s.pp + a.ampress));

} // Set Transformation matrix
setTrans();

// Ground Contact Location
n.gx = n.ax + n.rt * Math.sin(pitch);
n.gy = n.ayo;
n.gz = n.az - n.rt * Math.cos(pitch);

s.gx = s.ax + s.rt * Math.sin(pitch);
s.gy = s.ayo;
s.gz = s.az - s.rt * Math.cos(pitch);

p.gx = s.gx;
p.gy = -s.gy;
p.gz = s.gz;

// Axle Location
n.ay = n.ayo; // n.ax and n.az calculated in main loop
s.ay = s.ayo; // s.ax and s.az calculated in main loop

p.ax = s.ax;
p.ay = -s.ay;
p.az = s.az;

// Ground Axis Gear Loads
n.fyg = 0.0; // for symmetric solution, n.fxg and n.fzg calculated in main loop
s.fyg = 0.0; // for symmetric solution, s.fxg and s.fzg calculated in main loop

p.fxg = s.fxg;
p.fyg = s.fyg;
p.fzg = s.fzg;

// Body Axis Gear Loads
doTrans(0.0, 0.0, 0.0, n.fxg, n.fyg, n.fzg);

n.fx = grd[0];
n.fy = grd[1];
n.fz = grd[2];

doTrans(0.0, 0.0, 0.0, s.fxg, s.fyg, s.fzg);
s.fx = grd[0];
s.fy = grd[1];
s.fz = grd[2];

doTrans(0.0, 0.0, 0.0, p.fxg, p.fyg, p.fzg);
p.fx = grd[0];
p.fy = grd[1];
p.fz = grd[2];

// Nx
a.nx = 2. * s.fxg/a.gw; // check changed s.Fd to s.fxg

// compute ground height
// h = (ix - s.gx) * sina + (iz - s.gz) * cosa;
doTrans(n.gx, n.gy, n.gz, d.wltx, d.wlty, d.wltz);
d.twltd.setText (twodig.format(grd[2]));
doTrans(n.gx, n.gy, n.gz, d.vttx, d.vtty, d.vttz);
d.tvttt.setText (twodig.format(grd[2]));
doTrans(n.gx, n.gy, n.gz, d.rdx, d.rdy, d.rdz);
d.trdd.setText (twodig.format(grd[2]));
doTrans(n.gx, n.gy, n.gz, d.selx, d.sely, d.selz);
d.tsel.setText (twodig.format(grd[2]));

// calculate deck and roll angles
a.decka = pitch * 180.0 / 3.14159;
a.rola = 0.0; // for Symmetric Solution

// calculate % stroke
d.pstr = d.sstr;
d.nstrp = d.nstr / a.mmgs * 100.;
d.sstrp = d.sstr / a.mmgs * 100.;
d.pstrp = d.pstr / a.mmgs * 100.;

// output results
// angles
a.tdecka.setText(twodig.format(a.decka));
a.trolla.setText(twodig.format(a.roller));

// strokes
d.tpstr.setText(twodig.format(d.pstr));
d.tpstrp.setText(twodig.format(d.pstrp));
d.tnstr.setText(twodig.format(d.nstr));
d.tnstrp.setText(twodig.format(d.nstrp));
d.tsstr.setText(twodig.format(d.sstr));
d.tsstrp.setText(twodig.format(d.sstrp));

// Nose
n.tfxg.setText(twodig.format(n.fxg));
n.tfyg.setText(twodig.format(n.fyg));
n.tfzg.setText(twodig.format(n.fzg));
n.tfxb.setText(twodig.format(n.fxb));
n.tfyb.setText(twodig.format(n.fyb));
n.tfzb.setText(twodig.format(n.fzb));
n.tax.setText(twodig.format(n.ax));
n.tay.setText(twodig.format(n.ay));
n.taz.setText(twodig.format(n.az));
n.tgx.setText(twodig.format(n.gx));
n.tgy.setText(twodig.format(n.gy));
n.tgz.setText(twodig.format(n.gz));

// Port
p.tfxg.setText(twodig.format(p.fxg));
p.tfyg.setText(twodig.format(p.fyg));
p.tfzg.setText(twodig.format(p.fzg));
p.tfxb.setText(twodig.format(p.fxb));
p.tfyb.setText(twodig.format(p.fyb));
p.tfzb.setText(twodig.format(p.fzb));
p.tax.setText(twodig.format(p.ax));
p.tay.setText(twodig.format(p.ay));
p.taz.setText(twodig.format(p.az));
p.tgx.setText(twodig.format(p.gx));
p.tgy.setText(twodig.format(p.gy));
p.tgz.setText(twodig.format(p.gz));
// Starboard
s.tfxg.setText(twodig.format(s.fxg));
s.tfyg.setText(twodig.format(s.fyg));
s.tfzg.setText(twodig.format(s.fzg));

s.tfxb.setText(twodig.format(s.fxb));
s.tfyb.setText(twodig.format(s.fyb));
s.tfzb.setText(twodig.format(s.fzb));

s.tax.setText(twodig.format(s.ax));
s.tay.setText(twodig.format(s.ay));
s.taz.setText(twodig.format(s.az));

s.tgx.setText(twodig.format(s.gx));
s.tgy.setText(twodig.format(s.gy));
s.tgz.setText(twodig.format(s.gz));

// set stroke sliders
d.spstrp.setValue((int)Math.round(d.pstrp));
d.snstrp.setValue((int)Math.round(d.nstrp));
d.ssstrp.setValue((int)Math.round(d.sstrp));

public void setTrans()
{
  t[0][0] = Math.cos(-pitch)*Math.cos(yaw);
  t[0][1] = Math.cos(-pitch)*Math.sin(yaw);
  t[0][2] = -Math.sin(-pitch);

  t[1][0] = Math.sin(roll)*Math.sin(-pitch)*Math.cos(yaw)
    - Math.cos(roll)*Math.sin(yaw);
  t[1][1] = Math.sin(roll)*Math.sin(-pitch)*Math.sin(yaw)
    + Math.cos(roll)*Math.cos(yaw);
  t[1][2] = Math.sin(roll)*Math.cos(-pitch);

  t[2][0] = Math.cos(roll)*Math.sin(-pitch)*Math.cos(yaw)
    + Math.sin(roll)*Math.sin(yaw);
    - Math.sin(roll)*Math.cos(yaw);
  t[2][2] = Math.cos(roll)*Math.cos(-pitch);
}

public void doTrans( double x1, double y1, double z1,
                     double x2, double y2, double z2 )
{
  double grd0[] = { x1,y1,z1 };
  double acs[]   = { x2,y2,z2 };

  for (int j = 0; j <= 2; j++)
  {
    grd[j] = 0.0;
acs[j] = acs[j] - grd0[j];
for (int k = 0; k <= 2; k++)
{
    grd[j] = grd[j] + acs[k] * t[j][k];
}
//grd[j] = grd[j] - grd0[j];
}

private void getRemoteData( String acType, String ip )
{
    try
    {
        // name of remote server object bound to rmi registry
        String serverObjectName = "//" + ip + "/DataServer";

        // lookup ACDataServerImpl remote object in rmiregistry
        ACDataServer mydata = (ACDataServer) Naming.lookup(
            serverObjectName);

        // Get remote ACInfo object from Server
        ac = mydata.getAircraftInfo( acType );

        // Check HashMap Contents
        // System.err.println( ac.keySet() );
        // System.err.println( ac.values() );
        // System.err.println( ac.entrySet() );

        setAircraftValues( n, p, s, a, d, w, ac );
    }
    catch ( java.rmi.ConnectException ce )
    {
        System.err.println( "Connection to server failed. " +
            "Server may be temporarily unavailable." );
    }
    catch ( Exception e )
    {
        e.printStackTrace();
        System.exit(0);
    }
}

private void setAircraftValues(
    Gear cn, Gear cp, Gear cs,
    ACdata ca, StrokeDist cd, WtCGcanvas cw,
    HashMap map ) throws Exception
{
    // Get Classes
Class ccg = cn.getClass();
Class cca = ca.getClass();
Class ccd = cd.getClass();
Class ccw = cw.getClass();

// get the set of HashMap keys
Set s = map.keySet();
// initiate iterator using key set
Iterator i = s.iterator();
// loop on key set
while( i.hasNext() )
{
    // get the next key as a string
    String fieldName = (String)i.next();
    //System.err.println( fieldName );
    // get the value associated with the key
    double value = ((Double)map.get( fieldName )).doubleValue();

    // work with what instance
    if(fieldName.startsWith( "n." )) // nose gear = n
    {
        // field (variable) with same name as key (less the 2 char instance prefix)
        Field f = ccg.getField( fieldName.substring( 2 ) );
        // set the variable to value associated with the hashmap key
        f.setDouble( cn, value );
        // field (JTextField) with same name as key with t prefix less the 2 char instance prefix)
        f = ccg.getField( "t" + fieldName.substring( 2 ) );
        // get the JTextField
        JTextField jtf = (JTextField)f.get( cn );
        // set the JTextField to the Variable value
        jtf.setText( twodig.format( value ) );
        jtf.setHorizontalAlignment( javax.swing.SwingConstants.RIGHT);
        // if first set of aircraft data then add a listener
        if( firstAircraft ) jtf.addActionListener( this );
    }
    else if(fieldName.startsWith( "p." )) // port gear = p
    {
        Field f = ccg.getField( fieldName.substring( 2 ) );
        f.setDouble( cp, value );
        f = ccg.getField( "t" + fieldName.substring( 2 ) );
        JTextField jtf = (JTextField)f.get( cp );
        jtf.setText( twodig.format( value ) );
        jtf.setHorizontalAlignment( javax.swing.SwingConstants.RIGHT);
        if( firstAircraft ) jtf.addActionListener( this );
    }
else if (fieldName.startsWith("s.")) // starboard
gear = s
{
    Field f = ccg.getField(fieldName.substring(2));
f.setDouble(cs, value);
f = ccg.getField("t" + fieldName.substring(2));
JTextField jtf = (JTextField)f.get(cs);
jtf.setText(twodig.format(value));
jtf.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
    if( firstAircraft ) jtf.addActionListener(this);
}
else if (fieldName.startsWith("a.")) // ACdata
{
    Field f = cca.getField(fieldName.substring(2));
f.setDouble(ca, value);
f = cca.getField("t" + fieldName.substring(2));
JTextField jtf = (JTextField)f.get(ca);
jtf.setText(twodig.format(value));
jtf.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
    if( firstAircraft ) jtf.addActionListener(this);
}
else if (fieldName.startsWith("d.")) // StrokeDist
{
    Field f = ccd.getField(fieldName.substring(2));
f.setDouble(cd, value);
f = ccd.getField("t" + fieldName.substring(2));
JTextField jtf = (JTextField)f.get(cd);
jtf.setText(twodig.format(value));
jtf.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
    if( firstAircraft ) jtf.addActionListener(this);
}
else if (fieldName.startsWith("w.")) // WtCGcanvas
{
    Field f = ccw.getField(fieldName.substring(2));
f.setDouble(cw, value);
}
else // anything else
{
    JOptionPane.showMessageDialog(null,
"Key: " + fieldName + " 
with value: " + value + 
"has no Variable Match\nClient will now Stop");
System.exit(1);
}
private void updateVariables(Gear cn, Gear cp, Gear cs, ACdata ca, StrokeDist cd, WtCGcanvas cw, HashMap map) throws Exception
{

    // Get Classes
    Class ccg = cn.getClass();
    Class cca = ca.getClass();
    Class ccd = cd.getClass();
    Class ccw = cw.getClass();

    // get the set of HashMap keys
    Set s = map.keySet();
    // initiate iterator using key set
    Iterator i = s.iterator();
    // loop on key set
    while (i.hasNext())
    {

        // get the next key as a string
        String fieldName = (String)i.next();

        // work with what instance
        if (fieldName.startsWith("n.") ) // nose gear = n
        {
            Field f1 = ccg.getField( fieldName.substring(2) );
            Field f2 = ccg.getField( "t" + fieldName.substring(2) );
            JTextField jtf = (JTextField)f2.get( cn );
            double value = Double.parseDouble( jtf.getText() );
            f1.setDouble( cn, value );
        }
        else if (fieldName.startsWith("p.") ) // port gear = p
        {
            Field f1 = ccg.getField( fieldName.substring(2) );
            Field f2 = ccg.getField( "t" + fieldName.substring(2) );
            JTextField jtf = (JTextField)f2.get( cp );
            double value = Double.parseDouble( jtf.getText() );
            f1.setDouble( cp, value );
        }
        else if (fieldName.startsWith("s.") ) // starboard
        {
            Field f1 = ccg.getField( fieldName.substring(2) );
            Field f2 = ccg.getField( "t" + fieldName.substring(2) );
            JTextField jtf = (JTextField)f2.get( cs );
            double value = Double.parseDouble( jtf.getText() );
            f1.setDouble( cs, value );
        }
    }
}

}
else if(fieldName.startsWith("a.")) // ACData
{
    Field f1 = cca.getField(fieldName.substring(2));
    Field f2 = cca.getField("t" + fieldName.substring(2));
    JTextField jtf = (JTextField)f2.get("ca");
    double value = Double.parseDouble(jtf.getText());
    f1.setDouble("ca", value);
}
else if(fieldName.startsWith("d.")) // StrokeDist
{
    Field f1 = ccd.getField(fieldName.substring(2));
    Field f2 = ccd.getField("t" + fieldName.substring(2));
    JTextField jtf = (JTextField)f2.get("cd");
    double value = Double.parseDouble(jtf.getText());
    f1.setDouble("cd", value);
}
}

// If any JTextField is modified, all user defined variables are updated
public void actionPerformed(ActionEvent ev)
{
    try
    {
        // add check for valid entry (a number)
        ******************************
        updateVariables(n, p, s, a, d, w, ac);
        w.setZFGWpt(a.zfcg, a.zfgw);
        calResults();
    }
    catch (Exception e)
    {
        e.printStackTrace();
        System.exit(0);
    }
    // Get textfield which caused the event
    // JTextField field = (JTextField)ev.getSource();
    // field.setText((String)field.getName());
    // System.err.println(field + "||" + fs);
}

public void initresults()
{
    // Load Zero Fuel Envelope Polygon
w.setCanvas( w.deltaCG, w.deltaWT, w.offsetCG, w.offsetWT );

Polygon acZFE = new Polygon();
acZFE.addPoint( (int) w.ix( w.zfCG1 ), (int) w.iy( w.zfWT1 ) );
acZFE.addPoint( (int) w.ix( w.zfCG2 ), (int) w.iy( w.zfWT2 ) );
acZFE.addPoint( (int) w.ix( w.zfCG3 ), (int) w.iy( w.zfWT3 ) );
acZFE.addPoint( (int) w.ix( w.zfCG4 ), (int) w.iy( w.zfWT4 ) );
acZFE.addPoint( (int) w.ix( w.zfCG5 ), (int) w.iy( w.zfWT5 ) );
acZFE.addPoint( (int) w.ix( w.zfCG6 ), (int) w.iy( w.zfWT6 ) );
acZFE.addPoint( (int) w.ix( w.zfCG7 ), (int) w.iy( w.zfWT7 ) );
acZFE.addPoint( (int) w.ix( w.zfCG8 ), (int) w.iy( w.zfWT8 ) );
w.setZFGWE( acZFE );
w.setZFGWpt( a.zfcg, a.zfgw );

// Load Fuel Burn Curve
fuelburnCG[ 0 ] = w.fCG1;
fuelburnCG[ 1 ] = w.fCG2;
fuelburnCG[ 2 ] = w.fCG3;
fuelburnCG[ 3 ] = w.fCG4;
fuelburnCG[ 4 ] = w.fCG5;
fuelburnCG[ 5 ] = w.fCG6;
fuelburnCG[ 6 ] = w.fCG7;
fuelburnCG[ 7 ] = w.fCG8;
fuelburnCG[ 8 ] = w.fCG9;

fuelburnWT[ 0 ] = w.fWT1;
fuelburnWT[ 1 ] = w.fWT2;
fuelburnWT[ 2 ] = w.fWT3;
fuelburnWT[ 3 ] = w.fWT4;
fuelburnWT[ 4 ] = w.fWT5;
fuelburnWT[ 5 ] = w.fWT6;
fuelburnWT[ 6 ] = w.fWT7;
fuelburnWT[ 7 ] = w.fWT8;
fuelburnWT[ 8 ] = w.fWT9;

// set initial slider values
// strokes
//d.spstrup.setValue ( 100 );
//d.cnstrup.setValue ( 100 );
//d.ssstrup.setValue ( 100 );
// fuel
a.sfuel.setValue ( 0 );
a.tfuelcg.setText ("0.00");
// Calculate A/C dependent Geometry
// nose
n.theta = Math.atan((n.uxo - n.lxo)/(n.uzo - n.lzo));
// port
// star
s.rpa = Math.sqrt((s.px - s.axo) * (s.px - s.axo)+(s.py - s.ayo) * (s.py - s.ayo)+(s.pz - s.azo) * (s.pz - s.azo));
s.rpl = Math.sqrt((s.px - s.lxo) * (s.px - s.lxo)+(s.py - s.lyo) * (s.py - s.lyo)+(s.pz - s.lzo) * (s.pz - s.lzo));
s.rpu = Math.sqrt((s.px - s.uxo) * (s.px - s.uxo)+(s.py - s.uyo) * (s.py - s.uyo)+(s.pz - s.uzo) * (s.pz - s.uzo));
s.rul = Math.sqrt((s.lxo - s.uxo) * (s.lxo - s.uxo)+(s.lyo - s.uyo) * (s.lyo - s.uyo)+(s.lzo - s.uzo) * (s.lzo - s.uzo));
s.theta = Math.atan((s.azo - s.pz)/(s.axo - s.px));
s.thetaUPL = Math.acos(((s.uxo - s.px)*(s.lxo - s.pz)+(s.uzo - s.pz) * (s.lzo - s.pz))/(s.rpl * s.rpu));
class Gear extends javax.swing.JPanel {

    public Gear()
    {
        initComponents();
    }

    public void initComponents()
    {
        jTabbedPane = new javax.swing.JTabbedPane();
        jPanel1 = new javax.swing.JPanel();
        lgear = new javax.swing.JLabel();
        lfx = new javax.swing.JLabel();
        lfy = new javax.swing.JLabel();
        lfz = new javax.swing.JLabel();
        laxle = new javax.swing.JLabel();
        tax = new javax.swing.JTextField();
        tay = new javax.swing.JTextField();
        taz = new javax.swing.JTextField();
        lground = new javax.swing.JLabel();
        tgx = new javax.swing.JTextField();
        tgy = new javax.swing.JTextField();
        tgz = new javax.swing.JTextField();
        lforceg = new javax.swing.JLabel();
        tfxg = new javax.swing.JTextField();
        tfyg = new javax.swing.JTextField();
        tfzg = new javax.swing.JTextField();
        lforceb = new javax.swing.JLabel();
        tfxb = new javax.swing.JTextField();
        tfyb = new javax.swing.JTextField();
        tfzb = new javax.swing.JTextField();
        lopt1 = new javax.swing.JLabel();
        topt1 = new javax.swing.JTextField();
        lopt2 = new javax.swing.JLabel();
        topt2 = new javax.swing.JTextField();
        jPanel2 = new javax.swing.JPanel();
        lgear2 = new javax.swing.JLabel();
        lfs = new javax.swing.JLabel();
        lb1 = new javax.swing.JLabel();
        lw1 = new javax.swing.JLabel();
        laxle2 = new javax.swing.JLabel();
        taxo = new javax.swing.JTextField();
        tayo = new javax.swing.JTextField();
        tazo = new javax.swing.JTextField();
        lupper = new javax.swing.JLabel();
        tuxo = new javax.swing.JTextField();
        tuyo = new javax.swing.JTextField();
        tuzo = new javax.swing.JTextField();
        llower = new javax.swing.JLabel();
    }
tlxo = new javax.swing.JTextField ();
tlyo = new javax.swing.JTextField ();
tlzo = new javax.swing.JTextField ();
lpivot = new javax.swing.JLabel ();
tpx = new javax.swing.JTextField ();
tpy = new javax.swing.JTextField ();
tpz = new javax.swing.JTextField ();
lopt3 = new javax.swing.JLabel ();
topty3 = new javax.swing.JTextField ();
lopt4 = new javax.swing.JLabel ();
topty4 = new javax.swing.JTextField ();
jPanel3 = new javax.swing.JPanel ();
Ipo = new javax.swing.JLabel ();
tpo = new javax.swing.JTextField ();
lopt6 = new javax.swing.JLabel ();
topty6 = new javax.swing.JTextField ();
laa = new javax.swing.JLabel ();
taa = new javax.swing.JTextField ();
loa = new javax.swing.JLabel ();
toa = new javax.swing.JTextField ();
lvo = new javax.swing.JLabel ();
tvo = new javax.swing.JTextField ();
llov = new javax.swing.JLabel ();
tov = new javax.swing.JTextField ();
lrto = new javax.swing.JLabel ();
trto = new javax.swing.JTextField ();
lopt7 = new javax.swing.JLabel ();
topty7 = new javax.swing.JTextField ();
lkt = new javax.swing.JLabel ();
tkt = new javax.swing.JTextField ();
lopt8 = new javax.swing.JLabel ();
topty8 = new javax.swing.JTextField ();
lopt5 = new javax.swing.JLabel ();
topty5 = new javax.swing.JTextField ();
lopt9 = new javax.swing.JLabel ();
topty9 = new javax.swing.JTextField ();
setLayout (new java.awt.GridLayout (1, 1));
jTabbedPane.setBackground (java.awt.Color.lightGray);

jPanel1.setLayout (new java.awt.GridLayout (6, 4));
jPanel1.setBackground (java.awt.Color.cyan);

lgear.setText ("Title");
lgear.setForeground (java.awt.Color.blue);
lgear.setHorizontalAlignment (javax.swing.SwingConstants.CENTER);
lgear.setFont (new java.awt.Font ("Dialog", 1, 12));

jPanel1.add (lgear);

lfx.setText ("FS or FX");
lfx.setHorizontalAlignment
   (javax.swing.SwingConstants.CENTER);
   jPanel1.add (lfx);

lfy.setText ("BL or FY");
lfy.setHorizontalAlignment
   (javax.swing.SwingConstants.CENTER);
   jPanel1.add (lfy);

lfz.setText ("WL or FZ");
lfz.setHorizontalAlignment
   (javax.swing.SwingConstants.CENTER);
   jPanel1.add (lfz);

laxle.setText ("Axle");
laxle.setToolTipText("Location of Gear Axle");
   jPanel1.add (laxle);

tax.setText ("--");
tax.setEditable(false);
tax.setForeground(java.awt.Color.black);
   jPanel1.add (tax);

tay.setText ("--");
tay.setEditable(false);
   jPanel1.add (tay);

taz.setText ("--");
taz.setEditable(false);
   jPanel1.add (taz);

lground.setText("Ground");
lground.setToolTipText("Location of Ground Contact Point");
   jPanel1.add (lground);

tgx.setText("--");
tgx.setEditable(false);
   jPanel1.add (tgx);

tgy.setText("--");
tgy.setEditable(false);
   jPanel1.add (tgy);

tgz.setText("--");
tgz.setEditable(false);
   jPanel1.add (tgz);

lforceg.setText("Force(g)");
lforceg.setToolTipText("Gear Loads in Ground Coordinate System");
   jPanel1.add (lforceg);
tfxg.setText("---------");
tfxg.setEditable(false);
jPanel1.add(tfxg);

tfyg.setText("---------");
tfyg.setEditable(false);
jPanel1.add(tfyg);

tfzg.setText("---------");
tfzg.setEditable(false);
jPanel1.add(tfzg);

1forceb.setText("Force (b) ");
1forceb.setToolTipText("Gear Loads in Body Coordinate System");
jPanel1.add(1forceb);

tfxb.setText("---------");
tfxb.setEditable(false);
jPanel1.add(tfxb);

tfyb.setText("---------");
tfyb.setEditable(false);
jPanel1.add(tfyb);

tfzb.setText("---------");
tfzb.setEditable(false);
jPanel1.add(tfzb);

1opt1.setText("Option 1");
jPanel1.add(1opt1);

topt1.setText("---------");
jPanel1.add(topt1);

1opt2.setText("Option 2");
jPanel1.add(1opt2);

topt2.setText("---------");
jPanel1.add(topt2);

jTabbedPane1.addTab("Results", null, jPanel1, "Display Results");

ejPanel2.setLayout(new java.awt.GridLayout(6, 4));
ejPanel2.setBackground(java.awt.Color.pink);

1gear2.setText("Gear");
jPanel2.add(1gear2);

1fs.setText("FS(in)");
1fs.setHorizontalAlignment
(javax.swing.SwingConstants.CENTER);
jPanel2.add(lfs);

lbl.setText("BL(in)");
lbl.setHorizontalAlignment( javax.swing.SwingConstants.CENTER);
jPanel2.add(lbl);

lw1.setText("WL(in)");
lw1.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
jPanel2.add(lw1);

laxle2.setText("Axle");
laxle2.setBackground(java.awt.Color.pink);
jPanel2.add(laxle2);

taxo.setText("-
");
jPanel2.add(taxo);

tayo.setText("-
");
jPanel2.add(tayo);

tazo.setText("-
");
jPanel2.add(tazo);

lupper.setText("Upper Ref");
jPanel2.add(lupper);
	
tuxo.setText("-
");
jPanel2.add(tuxo);

tuyo.setText("-
");
jPanel2.add(tuyo);

tuzo.setText("-
");
jPanel2.add(tuzo);

llower.setText("Lower Ref");
jPanel2.add(llower);
	
tlxo.setText("-
");
jPanel2.add(tlxo);

tlyo.setText("-
");
jPanel2.add(tlyo);

tlzo.setText("-
");
jPanel2.add(tlzo);

lpivot.setText("Pivot");
jPanel2.add(lpivot);
tpx.setText("----------");
jPanel2.add(tpx);

tpy.setText("----------");
jPanel2.add(tpy);

tpz.setText("----------");
jPanel2.add(tpz);

lopt3.setText("Option 3");
jPanel2.add(lopt3);

topt3.setText("----------");
jPanel2.add(topt3);

lopt4.setText("Option 4");
lopt4.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
jPanel2.add(lopt4);

topt4.setText("----------");
jPanel2.add(topt4);

jTabbedPane1.addTab("Geometry", jPanel2);

jPanel3.setLayout(new java.awt.GridLayout(6, 4));
jPanel3.setBackground(java.awt.Color.green);

lpo.setText("Air Pressure");
jPanel3.add(lpo);

tpo.setText("----------");
jPanel3.add(tpo);

lopt6.setText("Option 6");
jPanel3.add(lopt6);

topt6.setText("----------");
jPanel3.add(topt6);

laa.setText("Air Area");
jPanel3.add(laa);

taa.setText("----------");
jPanel3.add(taa);

loa.setText("Oil Area");
jPanel3.add(loa);

toa.setText("----------");
jPanel3.add(toa);
lvo.setText("Air Volume");
jPanel3.add(lvo);

tvo.setText("--------");
jPanel3.add(tvo);

lov.setText("Oil Volume");
jPanel3.add(lov);

tov.setText("--------");
jPanel3.add(tov);

lrto.setText("Tire Radius");
jPanel3.add(lrto);

trto.setText("--------");
jPanel3.add(trto);

lopt7.setText("Option 7");
jPanel3.add(lopt7);

topt7.setText("--------");
jPanel3.add(topt7);

lkt.setText("Tire Stiffness");
jPanel3.add(lkt);

tkt.setText("--------");
jPanel3.add(tkt);

lopt8.setText("Option 8");
jPanel3.add(lopt8);

topt8.setText("--------");
jPanel3.add(topt8);

lopt5.setText("Option 5");
jPanel3.add(lopt5);

topt5.setText("--------");
jPanel3.add(topt5);

lopt9.setText("Option 9");
jPanel3.add(lopt9);

topt9.setText("--------");
jPanel3.add(topt9);

jTabbedPane1.addTab("Service", jPanel3);
add(jTabbedPane1);
public javax.swing.JTabbedPane jTabbedPane;
public javax.swing.JPanel jPanel1;
public javax.swing.JLabel lgear;
public javax.swing.JLabel lfx;
public javax.swing.JLabel lfy;
public javax.swing.JLabel lfx2;
public javax.swing.JLabel laxle;
public javax.swing.JTextField tax;
public javax.swing.JTextField tay;
public javax.swing.JTextField taz;
public javax.swing.JLabel Iforceg;
public javax.swing.JTextField tfxg;
public javax.swing.JTextField tfyg;
public javax.swing.JTextField tfzg;
public javax.swing.JLabel Iforceb;
public javax.swing.JTextField tfxb;
public javax.swing.JTextField tfyb;
public javax.swing.JTextField tfzb;
public javax.swing.JLabel loptl;
public javax.swing.JTextField toptl;
public javax.swing.JLabel lopt2;
public javax.swing.JTextField topt2;
public javax.swing.JLabel lopt3;
public javax.swing.JTextField topt3;
public javax.swing.JLabel lopt4;

}
public javax.swing.JTextField topt4;
public javax.swing.JPanel jPanel3;
public javax.swing.JLabel lpo;
public javax.swing.JTextField tpo;
public javax.swing.JLabel lopt6;
public javax.swing.JTextField topt6;
public javax.swing.JLabel laa;
public javax.swing.JTextField taa;
public javax.swing.JLabel lopt9;
public javax.swing.JTextField topt9;

// Results
public double fxg;
public double fyg;
public double fzg;
public double fxb;
public double fyb;
public double fzb;
public double opt1;
public double opt2;

// Geometry
public double axo;
public double ayo;
public doubleazo;
public double uxo;
public double uyo;
public double uzo;
public double lxo;
public double lyo;
public double lzo;
public double px;
public double py;
public double pz;
public double opt3;
public double opt4;
public double th3;
public double thax;
public double ax;
public double ay;
public double az;
public double gx;
public double gy;
public double gz;
public double fo;
public double pp;

public double theta;
public double thetaUPL;
public double rpa;
public double rpl;
public double rpu;
public double rul;

// service
public double po;
public double opt6;
public double aa;
public double oa;
public double vo;
public double ov;
public double rto;
public double rt;
public double opt7;
public double kt;
public double opt8;
public double opt5;
public double opt9;
}
class StrokeDist extends javax.swing.JPanel {

    public StrokeDist() {
        initComponents();
    }

    public void initComponents() {
        JTabbedPane jTabbedPane = new javax.swing.JTabbedPane();
        JPanel1 = new javax.swing.JPanel();
        JPanel2 = new javax.swing.JPanel();
        Ipstr = new javax.swing.JLabel();
        tpstr = new javax.swing.JTextField();
        Ipstrp = new javax.swing.JLabel();
        tpstrp = new javax.swing.JTextField();
        JPanel3 = new javax.swing.JPanel();
        Instr = new javax.swing.JLabel();
        tnstr = new javax.swing.JTextField();
        Instrp = new javax.swing.JLabel();
        tnstrp = new javax.swing.JTextField();
        JPanel4 = new javax.swing.JPanel();
        Isstr = new javax.swing.JLabel();
        tsstr = new javax.swing.JTextField();
        Isstrp = new javax.swing.JLabel();
        tsstrp = new javax.swing.JTextField();
        JPanel5 = new javax.swing.JPanel();
        JPanel6 = new javax.swing.JPanel();
        Iwlt = new javax.swing.JLabel();
        twltx = new javax.swing.JTextField();
        twltz = new javax.swing.JTextField();
        Ivtt = new javax.swing.JLabel();
        tvttx = new javax.swing.JTextField();
        tvtty = new javax.swing.JTextField();
        tvttz = new javax.swing.JTextField();
        Ird = new javax.swing.JLabel();
        trdx = new javax.swing.JTextField();
        trdy = new javax.swing.JTextField();
        trdz = new javax.swing.JTextField();
        Isel = new javax.swing.JLabel();
        tselx = new javax.swing.JTextField();
        tselz = new javax.swing.JTextField();
        jPanelV = new javax.swing.JPanel();
        Iwltd = new javax.swing.JLabel();
        twltd = new javax.swing.JTextField();
lvttd = new javax.swing.JLabel();
tvttd = new javax.swing.JTextField();
lrdd = new javax.swing.JLabel();
trdd = new javax.swing.JTextField();
lseid = new javax.swing.JLabel();
tseld = new javax.swing.JTextField();
setLayout (new java.awt.GridLayout (1, 1));

jPanel1.setLayout (new java.awt.GridLayout (2, 3));
jPanel2.setLayout (new java.awt.GridLayout (4, 1));

lpstr.setText ("Port Stoke (in)");
lpstr.setForeground (java.awt.Color.blue);
lpstr.setHorizontalAlignment
        (javax.swing.SwingConstants.CENTER);
jPanel2.add (lpstr);

tpstr.setText ("--------");
tpstr.setHorizontalAlignment
        (javax.swing.SwingConstants.CENTER);
jPanel2.add (tpstr);

lpstrp.setText ("Port Stoke (%)");
lpstrp.setForeground (java.awt.Color.blue);
lpstrp.setHorizontalAlignment
        (javax.swing.SwingConstants.CENTER);
jPanel2.add (lpstrp);

tpstrp.setText ("--------");
tpstrp.setHorizontalAlignment
        (javax.swing.SwingConstants.CENTER);
jPanel2.add (tpstrp);

jPanel1.add (jPanel2);

jPanel3.setLayout (new java.awt.GridLayout (4, 1));

lnstr.setText ("Nose Stroke (in)");
lnstr.setForeground (java.awt.Color.blue);
lnstr.setHorizontalAlignment
        (javax.swing.SwingConstants.CENTER);
jPanel3.add (lnstr);

tnstr.setText ("--------");	nstr.setHorizontalAlignment
        (javax.swing.SwingConstants.CENTER);
jPanel3.add (tnstr);

lnstrp.setText ("Nose Stroke (%)");
lnstrp.setForeground (java.awt.Color.blue);
lnstrp.setHorizontalAlignment
   (javax.swing.SwingConstants.CENTER);
jpnl3.add (lnstrp);

tnstrp.setText ("----------");
tnstrp.setHorizontalAlignment
   (javax.swing.SwingConstants.CENTER);
jpnl3.add (tnstrp);

jpnl1.add (jpnl3);

jpnl4.setLayout (new java.awt.GridLayout (4, 1));

lsstr.setText ("Star Stroke (in)");
lsstr.setForeground (java.awt.Color.blue);
lsstr.setHorizontalAlignment
   (javax.swing.SwingConstants.CENTER);
jpnl4.add (lsstr);

tssstr.setText ("----------");
tssstr.setHorizontalAlignment
   (javax.swing.SwingConstants.CENTER);
jpnl4.add (tssstr);

lsstrp.setText ("Star Stroke (%)");
lsstrp.setForeground (java.awt.Color.blue);
lsstrp.setHorizontalAlignment
   (javax.swing.SwingConstants.CENTER);
jpnl4.add (lsstrp);

tssstrp.setText ("----------");
tssstrp.setHorizontalAlignment
   (javax.swing.SwingConstants.CENTER);
jpnl4.add (tssstrp);

jpnl1.add (jpnl4);

spstrp.setInverted (true);
spstrp.setMinorTickSpacing (5);
spstrp.setPaintLabels (true);
spstrp.setPaintTicks (true);
spstrp.setOrientation
   (javax.swing.SwingConstants.VERTICAL);
spstrp.setMinimum (70);
spstrp.setMajorTickSpacing (5);
spstrp.setForeground (java.awt.Color.blue);
spstrp.setBackgroundColor (java.awt.Color.cyan);
spstrp.setValue (100);
jPanell.add (spstrp);

snstrp.setInverted (true);
snstrp.setMinorTickSpacing (20);
snstrp.setPaintLabels (true);
snstrp.setPaintTicks (true);
snstrp.setOrientation
   (javax.swing.SwingConstants.VERTICAL);
snstrp.setMajorTickSpacing (20);
snstrp.setBackground (java.awt.Color.cyan);
snstrp.setValue (100);

jPanell.add (snstrp);

ssstrp.setInverted (true);
ssstrp.setMinorTickSpacing (5);
ssstrp.setPaintLabels (true);
ssstrp.setPaintTicks (true);
ssstrp.setOrientation
   (javax.swing.SwingConstants.VERTICAL);
ssstrp.setMinimum (70);
ssstrp.setMajorTickSpacing (5);
ssstrp setBackground (java.awt.Color.cyan);
ssstrp.setValue (100);

jPanell.add (ssstrp);

jTabbedPane1.addTab ("Gear Stoke", null, jPanell,  
   "Display of Gear Stroke");

jPanel5.setLayout (new java.awt.GridLayout (2, 1));
jPanel6.setLayout (new java.awt.GridLayout (4, 4));

lwlt.setText ("Winglet Tip");
jPanel6.add (lwlt);

twltx.setText ("--------");
twltx.setHorizontalAlignment
   (javax.swing.SwingConstants.RIGHT);
jPanel6.add (twltx);

twltty.setText ("--------");
twltty.setHorizontalAlignment
   (javax.swing.SwingConstants.RIGHT);
jPanel6.add (twltty);

twlttz.setText ("--------");
twlttz.setHorizontalAlignment
   (javax.swing.SwingConstants.RIGHT);
jPanel6.add (twlttz);
lvtt.setText("VT Tip");
jPanel6.add(lvtt);

tvttx.setText("-\--\--\--\--\--\--\--\--\--");
tvttx.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
jPanel6.add(tvttx);

tvttty.setText("-\--\--\--\--\--\--\--\--\--");
tvttty.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
jPanel6.add(tvttty);

tvtttz.setText("-\--\--\--\--\--\--\--\--\--");
tvtttz.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
jPanel6.add(tvtttz);

1rd.setText("Radome");
jPanel6.add(1rd);

trdx.setText("-\--\--\--\--\--\--\--\--\--");
trdx.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
jPanel6.add(trdx);

trdy.setText("-\--\--\--\--\--\--\--\--\--");
trdy.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
jPanel6.add(trdy);

trdz.setText("-\--\--\--\--\--\--\--\--\--");
trdz.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
jPanel6.add(trdz);

lsel.setText("Selected");
jPanel6.add(lsel);

tselx.setText("-\--\--\--\--\--\--\--\--\--");
tselx.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
jPanel6.add(tselx);

tsely.setText("-\--\--\--\--\--\--\--\--\--");
tsely.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
jPanel6.add(tsely);

tselz.setText("-\--\--\--\--\--\--\--\--\--");
tselz.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
jPanel6.add (tsselz);
jPanel5.add (jPanel6);

jPanel7.setLayout (new java.awt.GridLayout (4, 2));

lwltld.setText ("Ground to Winglet Tip");
lwltld.setHorizontalAlignment (javax.swing.SwingConstants.RIGHT);
jPanel7.add (lwltld);

twltld.setText ("---------");
twltld.setHorizontalAlignment (javax.swing.SwingConstants.RIGHT);
jPanel7.add (twltld);

lvttd.setText ("to VT Tip");
lvttd.setHorizontalAlignment (javax.swing.SwingConstants.RIGHT);
jPanel7.add (lvttd);

tvtttd.setText ("---------");
tvtttd.setHorizontalAlignment (javax.swing.SwingConstants.RIGHT);
jPanel7.add (tvtttd);

lrdd.setText ("to Radome");
lrdd.setHorizontalAlignment (javax.swing.SwingConstants.RIGHT);
jPanel7.add (lrdd);

trdd.setText ("---------");
trdd.setHorizontalAlignment (javax.swing.SwingConstants.RIGHT);
jPanel7.add (trdd);

lseld.setText ("to Selected");
lseld.setHorizontalAlignment (javax.swing.SwingConstants.RIGHT);
jPanel7.add (lseld);

tseld.setText ("---------");
tseld.setHorizontalAlignment (javax.swing.SwingConstants.RIGHT);
jPanel7.add (tseld);

jPanel5.add (jPanel7);

jTabbedPane1.addTab ("Distance to Ground", jPanel5);

add (jTabbedPane1);
public javax.swing.JTabbedPane jTabbedPane;
public javax.swing.JPanel jPanel1;
public javax.swing.JPanel jPanel2;
public javax.swing.JLabel lpstr;
public javax.swing.JTextField tpstr;
public javax.swing.JLabel lpstrp;
public javax.swing.JTextField tpstrp;
public javax.swing.JPanel jPanel3;
public javax.swing.JLabel Instr;
public javax.swing.JTextField tnstr;
public javax.swing.JLabel Instrp;
public javax.swing.JTextField tnstrp;
public javax.swing.JPanel jPanel6;
public javax.swing.JLabel Iwlt;
public javax.swing.JTextField twltx;
public javax.swing.JTextField twlty;
public javax.swing.JTextField twltz;
public javax.swing.JLabel Ivtt;
public javax.swing.JTextField tvttx;
public javax.swing.JTextField tvtty;
public javax.swing.JTextField tvttz;
public javax.swing.JLabel Ird;
public javax.swing.JTextField trdx;
public javax.swing.JTextField trdy;
public javax.swing.JTextField trdz;
public javax.swing.JLabel Isel;
public javax.swing.JTextField tselx;
public javax.swing.JTextField tsely;
public javax.swing.JTextField tselz;
public javax.swing.JPanel jPanelV;
public javax.swing.JLabel Iwltd;
public javax.swing.JTextField twltd;
public javax.swing.JLabel Ivttd;
public javax.swing.JTextField tvttd;
public javax.swing.JLabel Irdd;
public javax.swing.JTextField trdd;
public javax.swing.JLabel Iseld;
public javax.swing.JTextField tseld;

public double pstr;
public double pstrp;
public double nstr;
public double nstrp;
public double sstr;
public double sstrp;
public double wltx;
public double wlty;
public double wltz;
public double vttx;
public double vtty;
public double vttz;
public double rdx;
public double rdy;
public double rdz;
public double selx;
public double sely;
public double selz;
public double wltd;
public double vttd;
public double rdd;
public double seld;
WtCGcanvas.java

// class WtCGcanvas shows the ZFGW Envelope of the Selected A/C
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import java.text.DecimalFormat;

class WtCGcanvas extends JPanel
{
    // Aircraft weight-CG plot data
    public double mac;
    public double macLE;
    public double offsetCG = 0.0;
    public double offsetWT = 0.0;
    public double deltaCG = 20.0;
    public double deltaWT = 20000.0;
    // Zero Fuel Envelope CG Points
    public double zfCG1 = 2.0;
    public double zfCG2 = 18.0;
    public double zfCG3 = 18.0;
    public double zfCG4 = 2.0;
    public double zfCG5 = 2.0;
    public double zfCG6;
    public double zfCG7;
    public double zfCG8;
    // Zero Fuel Envelope Weight Points
    public double zfWT1 = 18000.0;
    public double zfWT2 = 18000.0;
    public double zfWT3 = 2000.0;
    public double zfWT4 = 2000.0;
    public double zfWT5 = 18000.0;
    public double zfWT6;
    public double zfWT7;
    public double zfWT8;
    // Fuel Burn Data CG(WT)
    public double fCG1;
    public double fCG2;
    public double fCG3;
    public double fCG4;
    public double fCG5;
    public double fCG6;
    public double fCG7;
    public double fCG8;
    public double fCG9;
    public double fWT1;
    public double fWT2;
    public double fWT3;
    public double fWT4;
    public double fWT5;
public double fWT6;
public double fWT7;
public double fWT8;
public double fWT9;

public int ii = 0;
public int maxX;
public int maxY;
public double pixelWidth;
public double pixelHeight;
public Polygon wtcgEnv = new Polygon();

public Point pt = new Point();
public double cg = 0.0;
public double wt = 0.0;
ACdata a;
StaticGear s;

double fx(int x){return x * pixelWidth + offsetCG;}
double fy(int y){return (maxY - y) * pixelHeight +
    offsetWT;}
long ix(double x){return Math.round((x -
    offsetCG)/pixelWidth);}
long iy(double y){return maxY - Math.round((y -
    offsetWT)/pixelHeight);}

public WtCGcanvas(ACdata a, StaticGear s)
{
    this.a = a;
    this.s = s;

    Dimension size = getSize();
    maxX = size.width - 1;
    maxY = size.height - 1;
    pixelWidth = deltaCG/maxX;
    pixelHeight = deltaWT/maxY;

    // initialize polygon
    wtcgEnv.addPoint( (int) ix( fCG1 ), (int) iy( fWT1 ) );
    wtcgEnv.addPoint( (int) ix( fCG1 ), (int) iy( fWT2 ) );
    wtcgEnv.addPoint( (int) ix( fCG1 ), (int) iy( fWT3 ) );
    wtcgEnv.addPoint( (int) ix( fCG1 ), (int) iy( fWT4 ) );
    wtcgEnv.addPoint( (int) ix( fCG1 ), (int) iy( fWT5 ) );
    repaint();
}

public void setCanvas( double deltaCG, double deltaWT,
    double offsetCG, double offsetWT )
{
    this.deltaCG = deltaCG;
    this.deltaWT = deltaWT;
this.offsetCG = offsetCG;
this.offsetWT = offsetWT;

Dimension size = getSize();
maxX = size.width - 1;
maxY = size.height - 1;
pixelWidth = deltaCG/maxX;
pixelHeight = deltaWT/maxY;
}

public void setZFGWE( Polygon wtcgEnv )
{
    this.wtcgEnv = wtcgEnv;
    repaint();
}

public void setZFGWpt( double cg, double wt )
{
    this.cg = cg;
    this.wt = wt;
    pt.x = (int) ix(cg);
    pt.y = (int) iy(wt);
    repaint();
}

public void paintComponent(Graphics g)
{
    super.paintComponent(g);

    g.setColor(Color.black);
g.drawString("Aircraft Zero Fuel Gross Weight Envelope", 20, 20);
g.drawString("WT", 5, maxY/2);
g.drawString("CG(%)", maxX/2, maxY-20);

g.setColor(Color.cyan);
g.fillPolygon( wtcgEnv );
g.setColor(Color.blue);
g.drawPolygon( wtcgEnv );

g.drawLine(pt.x,pt.y - 10,pt.x,pt.y+10);
g.drawLine(pt.x - 10,pt.y,pt.x+10,pt.y);

addMouseMotionListener(new MouseMotionAdapter()
{
    public void mouseDragged(MouseEvent evt)
    {
        Graphics g = getGraphics();
        pt.x = evt.getX();
        pt.y = evt.getY();
    }
});
pt.y = evt.getY();
cg = fx(pt.x);
wt = fy(pt.y);

g.setColor(Color.red);
g.drawLine(pt.x, pt.y - 8, pt.x, pt.y + 8);
g.drawLine(pt.x - 8, pt.y, pt.x + 8, pt.y);
// repaint();

s.setwtcg(wt, cg);

ii = ii + 1;
// System.out.println("Call calResults from drag" + ii);
s.calResults();

});
}

addMouseListener(new MouseAdapter()
{
    public void mouseReleased(MouseEvent evt)
    {
        Graphics g = getGraphics();

        pt.x = evt.getX();
        pt.y = evt.getY();
        cg = fx(pt.x);
        wt = fy(pt.y);

        repaint();

        s.setwtcg(wt, cg);

        ii = ii + 1;
        // System.out.println("Call calResults from Mouse Release" + ii);
        s.calResults();
    }
});

addMouseListener(new MouseAdapter()
{
    public void mousePressed(MouseEvent evt)
    {
        repaint();
    }
});

BIBLIOGRAPHY


Leen Ammeraa, Computer Graphics for Java Programers, John Wiley and Sons Ltd 1998