Flying Tiger Phase I

6-DOF Aircraft Stick-to-Tail Flight Simulator

Team Foxtrot Tango
Geoff Carlson, Corey Engelman, Mark Kerbel,
Brett Snare, Conlan Wesson

Project Sponsor
MOOG Inc.

Mechanical Engineering Sponsors
Heather Hussain
Dr. Jason Kolodziej
Dr. Agamemnon L. Crassidis

Faculty Coach
Thomas J. Reichlmayr

Project Year
2012
# Table of Contents

Project Overview ................................................................................................................. 3

Basic Requirements ............................................................................................................... 3

Constraints ............................................................................................................................. 8

Development Process ......................................................................................................... 8

Project Schedule: Planned and Actual .................................................................................. 9

System Design ....................................................................................................................... 9

Process and Product Metrics ............................................................................................... 11

Product State at Time of Delivery ....................................................................................... 13

Project Reflection ................................................................................................................ 13

References ............................................................................................................................ Error! Bookmark not defined.

APPENDICES ....................................................................................................................... 15

Appendix A – Gauge information .......................................................................................... 15

Interior Display – Gauges ....................................................................................................... 15

Appendix B – ARINC Library information .......................................................................... 20

Class Roles & Responsibilities .............................................................................................. 20

Function Definitions ............................................................................................................. 22

  ft_board_manager .................................................................................................................. 22

  ft_board_util ........................................................................................................................ 24

  ft_flightgear_properties ...................................................................................................... 26

  ft_arinc_dictionary .............................................................................................................. 30

  ft_channel_monitor ............................................................................................................ 36

  ft_arinc_transmitter ............................................................................................................ 39

Class Diagram ....................................................................................................................... 41

Adding/Removing Variables ................................................................................................. 41

Appendix C – User Setup ...................................................................................................... 43

Installing ................................................................................................................................. 43

Executables ............................................................................................................................ 43

Configuration .......................................................................................................................... 43
Project Overview

Moog Inc. is a corporation that designs and manufactures fluid control systems for various industries worldwide. The aviation branch of East Aurora, NY will be sponsoring a multidisciplinary senior project at RIT in which students will be developing a flight simulator. The simulator will be used for customer demonstrations to help sell Moog systems.

Our software engineering team will be responsible for the visual content displayed in the flight simulator. Our system will receive information from a system called the dSPACE controller and will generate visual feedback in the form of a virtual cockpit window and instrument display. The dSPACE controller will be developed by Moog and acts as a façade between the hardware inputs and our software package. The graphical output will be rendered using flight simulation software that is already available. The instrumentation display will be created based off of the actual physical instruments found in the aircraft being converted into a flight simulator.

Context Diagram

**Basic Requirements**

3D Environment
- Curved screen, geometric correction
- Simulation Synchronization Warning

Instrument Panel
- Air Speed Indicator
- Artificial Horizon
- Vertical Speed Indicator
- Turn & Bank Indicator
- Heading Indicator
- Altimeter
- Stall Warning
- Multiple displays
  - Two, mirrored instrument panels
  - Multiple projectors for environment
- ARINC-429 compatible
- Customizable interfaces
- Multiple input methods
- dSPACE flight dynamics model

<table>
<thead>
<tr>
<th>User Story</th>
<th>Task</th>
<th>Business Value</th>
<th>Story Points</th>
<th>Acceptance Criteria</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>As a pilot I want to be warned when the plane is about to stall so that it</td>
<td>1. Implement stall warning light on flying wall.</td>
<td>Med</td>
<td>3</td>
<td>The stall warning light is displayed when the plane is within a &lt;TBD&gt; threshold of</td>
<td>Done</td>
</tr>
<tr>
<td>does not stall.</td>
<td></td>
<td></td>
<td></td>
<td>stalling.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2. Receive stall warning from ARINC.</td>
<td></td>
<td>2</td>
<td></td>
<td>Done</td>
</tr>
<tr>
<td></td>
<td>3. Update warning light accordingly.</td>
<td></td>
<td>2</td>
<td></td>
<td>Assigned</td>
</tr>
<tr>
<td>As a pilot I want to be warned when the 3D environment is out of sync</td>
<td>1. Change on screen display to low-g setting when out-</td>
<td>Low</td>
<td>7</td>
<td>The sync warning light is displayed when the 3D environment and motion table are</td>
<td>Done</td>
</tr>
<tr>
<td>with the motion table so that I am aware of it being out of sync.</td>
<td>of-sync.</td>
<td></td>
<td></td>
<td>out of sync.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1. Create &quot;flying wall&quot; aircraft model.</td>
<td>High</td>
<td>3</td>
<td>All the flight instrumentation is displayed on a separate display than the environment.</td>
<td>Done</td>
</tr>
<tr>
<td>As a pilot I want a separate instruments display so that the environment</td>
<td>1. Implement stall warning light on flying wall.</td>
<td>Med</td>
<td>3</td>
<td>All the flight instrumentation is displayed on a separate display than the environment.</td>
<td>Done</td>
</tr>
<tr>
<td>display is not obstructed. (originally estimated as 7SP)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task</td>
<td>Complexity</td>
<td>Status</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>------------</td>
<td>---------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. Set first camera to point out at the environment.</td>
<td>3</td>
<td>Done</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. Set second camera to point at the wall.</td>
<td>2</td>
<td>Done</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4. Changing views does not change instrument display.</td>
<td>3</td>
<td>Done</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>As a pilot I want an instrument display so that I don't crash the plane.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. Indicated air speed gauge.</td>
<td>High</td>
<td>The six instruments are displayed and update properly.</td>
<td>Done</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. Artificial horizon dial.</td>
<td>3</td>
<td>Done</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. Heading Indicator.</td>
<td>3</td>
<td>Done</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4. Climb/sink rate indicator.</td>
<td>3</td>
<td>Done</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5. Altimeter.</td>
<td>3</td>
<td>Done</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6. Turn coordinator.</td>
<td>4</td>
<td>Done</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>As a pilot I want the controls in the cockpit to control the 3D simulation on the screen so that I can experience simulated</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. Implement ARINC reader.</td>
<td>Med</td>
<td>The 3D environment reacts as expected to the cockpit controls.</td>
<td>Done</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
As a pilot I want the display to refresh at least 60Hz so that my experience is not lagging.

<table>
<thead>
<tr>
<th>Task</th>
<th>Status</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2. Implement ARINC parser</td>
<td>5</td>
<td>Done</td>
</tr>
<tr>
<td>3. Setup FlightGear build environment in Windows</td>
<td>10</td>
<td>Cancelled</td>
</tr>
<tr>
<td>4. Embed ARINC FDM into FlightGear</td>
<td>4</td>
<td>Cancelled</td>
</tr>
<tr>
<td>5. Send data from ARINC reader to FlightGear via the remote interface</td>
<td>3</td>
<td>To verify</td>
</tr>
<tr>
<td>6. Interface dSPACE computer w/ ARINC transmitter from item 8</td>
<td>4</td>
<td>Assigned</td>
</tr>
<tr>
<td>7. Make ARINC parser configurable using either a tutorial or UI</td>
<td>4</td>
<td>Cancelled</td>
</tr>
<tr>
<td>8. Implement ARINC transmitter</td>
<td>3</td>
<td>Done</td>
</tr>
</tbody>
</table>

As a pilot I want the display to refresh at least 60Hz so that my experience is not lagging.
<table>
<thead>
<tr>
<th>As an administrator I want to be able to read flight data from a file so that the source and format can be changed.</th>
<th>Low</th>
<th>4</th>
<th>The data is read properly and the format can be easily changed.</th>
<th>Done</th>
</tr>
</thead>
<tbody>
<tr>
<td>As an administrator I want a tutorial for creating/customizing the six main instruments.</td>
<td>1. Determine the difference between updating gauges through nasal and XML.</td>
<td>TBD</td>
<td>2</td>
<td>The customer is able to understand the tutorial and create new gauges.</td>
</tr>
<tr>
<td></td>
<td>2. Write tutorials for creating instruments.</td>
<td>3</td>
<td></td>
<td>Done</td>
</tr>
<tr>
<td>As the Software Engineering department I would like a project poster to visualize the project.</td>
<td>1. Attend project poster meeting (4/3).</td>
<td>High</td>
<td>1</td>
<td>The project poster has been submitted to be printed.</td>
</tr>
<tr>
<td></td>
<td>2. Create the project poster concept (week 5).</td>
<td>2</td>
<td></td>
<td>Done</td>
</tr>
<tr>
<td></td>
<td>3. Using provided template(s) create preliminary project poster (week 6).</td>
<td>4</td>
<td></td>
<td>Done</td>
</tr>
<tr>
<td></td>
<td>4. Create final project poster (4/25).</td>
<td>4</td>
<td></td>
<td>To verify</td>
</tr>
<tr>
<td>As the Software Engineering department I would like a technical report to determine if the students may graduate.</td>
<td>1. Create technical report outline (week 8).</td>
<td>High</td>
<td>4</td>
<td>The technical report is complete and submitted.</td>
</tr>
</tbody>
</table>
2. Draft the technical report (week 10).

3. Write final technical report (week 11).

As a pilot I want the 3D environment display to be geometrically corrected to match the screen shape.

1. Install PixelWix software on the FlightGear computer. High

2. Run FlightGear through PixelWix.

3. Perform all tests on the system with PixelWix installed.

Constraints

The project involved several constraints. From the beginning of the project, dSPACE was required for the flight dynamics model. This meant that the flight simulator that was to be developed would need to interface with dSPACE rather than using a built-in flight dynamics model. Additionally, the software was required to be projected on a curved screen with geometric correction. Early in the development process, FlightGear was added as a constraint. FlightGear was selected as the flight simulator that would be used because of its open-source nature. After much discussion, ARINC-429 was settled on to provide the connection between the dSPACE computer and the FlightGear computer. ARINC-429 was chosen because it is used in all modern aircraft to facilitate communication between the various subsystems. It will also allow Moog to connect additional equipment to the communications bus for testing and demonstration purposes.

Development Process

This project used a slightly modified scrum methodology. The schedule consisted of six two-week sprints with a standup meeting twice a week and a sprint retrospective and planning on Tuesdays between sprints. The sponsor did not mandate or require approval of the process used.
The sponsor was invited to all regularly scheduled meetings. Corey Engelman, being the elected team leader, was appointed as the scrum master.

Project Schedule: Planned and Actual

The final schedule is shown below. Sprint 1 was initially scheduled to start sooner, however it was pushed back after encountering problems with initial setup work being done during Sprint 0. The other sprints were likewise pushed back to match. This made the Release Sprint shorter than initially intended. Further complications were encountered with Sprint 3 overlapping with the scheduled Winter/Spring break. Rather than end the sprint during the break, the sprint was extended one week, and beginning Sprint 4 following the first regularly scheduled meeting.

System Design

This diagram shows our most significant design decision. The blue portion shows how we could have made the ARINC-429 software internal to FlightGear and recompiled FlightGear. In other words, FlightGear would access the hardware directly to update itself. The green portion shows our selected solution, in which we make the ARINC-429 software external to FlightGear, and send data to FlightGear.
The deployment view below shows how our software interacts with the other pieces in our hardware environment. The boxes with depth show physical separation, while other boxes show components. The main point of this diagram is to show cockpit sensors sending signals to the dSPACE machine, where it outputs flight simulation data to our ARINC-429 library (a copy of which is on both the dSPACE and FlightGear computers). The library then handles encoding data and putting it on the hardware on the dSPACE machine. This sends it to the hardware on the FlightGear machine. The copy of the ARINC-429 library on the receive side then decodes all the data coming off the hardware and sends it to FlightGear view sockets. Our XML data specification and configuration files are also on this machine and are read by FlightGear when it first runs.
Process and Product Metrics

The team tracked story points throughout the development process. After initial planning during Sprint 0, there were 59 story points in total. At various points, additional points were added to the backlog as additional requirements were added. Some requirements were eventually canceled, the points associated with that tasks are not included in the data shown here. The number of story points assigned to each task was occasionally adjusted as the team began to suspect a task would be easier or harder. As shown in the graph below, after a slow start, the team began to average 13-14 story points per sprint, well above the ideal velocity of 8-9 story points per sprint.
<table>
<thead>
<tr>
<th>Sprint</th>
<th>Pts. Remaining</th>
<th>Points Complete</th>
<th>Ideal Points</th>
<th>Velocity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sprint 0 (S0)</td>
<td>59</td>
<td>0</td>
<td>59</td>
<td>n/a</td>
</tr>
<tr>
<td>Sprint 1 (S1)</td>
<td>59</td>
<td>4</td>
<td>50.5</td>
<td>4</td>
</tr>
<tr>
<td>Sprint 2 (S2)</td>
<td>60</td>
<td>11</td>
<td>42</td>
<td>9.5</td>
</tr>
<tr>
<td>Sprint 3 (S3)</td>
<td>50</td>
<td>15</td>
<td>33.5</td>
<td>10</td>
</tr>
</tbody>
</table>
### Product State at Time of Delivery

The product is not complete due to several setbacks the entire Flying Tiger project experienced. Currently, the project consists of two key aspects: the FlightGear configuration, and the ARINC library. The configuration files include everything required to run FlightGear with the large 3D environment display and instrument panel. The ARINC library reads flight data from a file on one computer, transmits it across the ARINC bus, and passes the data to FlightGear, which then renders the flight accordingly. The data is read and transmitted at a rate of 64Hz.

Additional work will be required to interface the transmitting side with the dSPACE flight dynamics model. There is a faulty ARINC cable which causes all the bits transmitted to be inverted, there is a workaround in-place to compensate for this issue. That workaround will need to be removed if a new ARINC cable is used. The workaround exists in the `ft_channel_monitor.cpp` class. There is a line that says “if( is_inverting_bits() )”.

The latitude and longitude labels used in the ARINC specification provide insufficient resolution for smoothly rendering a simulation. To circumvent this problem, the project includes support for custom ARINC labels which carry extended precision on top of the standard latitude and longitude. The same is true for the roll and pitch.

### Project Reflection

Our team chose to use the Scrum framework to guide our project. We chose this path after concluding that the largest risks our team faced would be unknown requirements and an unclear vision of what the final product will be for the team. This worked very well for intercepting changing and brand new requirements well into the project timeline. In the future, software engineer teams might be encouraged to teach their project sponsors about the development
process they are using. Our sponsors were not aware of the Scrum framework and may have become frustrated with how our team operated. For example, a new requirement may have been introduced mid-sprint that the team was unable to address immediately. The agile methodology requires frequent communication with sponsors which is something the team had in the beginning of the project but found to be lacking towards the conclusion. It would be better if there was better integration between the College of Engineering’s teams curriculum and our departments.
APPENDICES

Appendix A – Gauge information

Interior Display – Gauges

Gauges in order: Brake light, gear light, stall warning light, airspeed indicator, artificial horizon, altimeter, turn and bank indicator, directional gyro, and vertical speed indicator

The interior display is created using XML files to define the placement, and movement of images that are desired for display. If you are unfamiliar with XML, a good resource is available from w3schools (http://www.w3schools.com/xml/). Our solution uses two dimensional images but it is possible to use three dimensional instruments (http://wiki.flightgear.org/Creating_instruments_for_FG).

The base XML file that defines the instrument panel is called ft-panel.xml (FlightGear\data\Aircraft\FlyingTiger\ft-panel.xml). Instruments are added to the display between the <instruments> tags. For example, we have included a Chronometer for aesthetic purposes:

    <instrument include="..../Instruments/clock.xml">
<name>Chronometer</name>
<x>177</x>
<y>316</y>
<w>74</w>
<h>74</h>
</instrument>

The include attribute points to the location of the gauge’s XML file. The x and y tags define where on the display the instrument is located and the w and h tags define how wide and high the instrument is. You can edit the XML file while running the simulation to see how removing different instruments affects the display in real-time. Use Shift-F3 to refresh the instrument panel to see how editing the XML file operates.

The scope of the Flying Tiger project included creating the instruments highlighted below based on existing ones available in the software. There are numerous instruments available that can be added to the interior display that have already been created and are available with every FlightGear install (FlightGear\data\Aircraft\Instruments). Each instrument has image files associated with them that can be edited as needed (FlightGear\data\Aircraft\Instruments\Textures).

Each gauge has the same XML layout. Each image that makes up the instrument is added as a <layer> ordered from bottom to top. Each <layer> is added between the <layers> tag. Each layer has the following tags:

- <name> - Name of the instrument
- <texture> - Adds image of this layer
  o <path> - Location of the image
  o <x1>, <y1>, <x2>, <y2> - Coordinate location of where to capture the image as a percentage where 1.0 = 100%.

  <x1>0.0</x1>
  <y1>0.0</y1>
  <x2>1.0</x2>
  <y2>1.0</y2>

  The configuration above grabs all of the texture

- <transformations> - All animation definitions are included between this tag
  o <transformation> - Adds an animation
    ▪ <type> - Type of animation. The documentation of animations is out of date but are defined on the FlightGear Wiki (http://wiki.flightgear.org/Howto:_Animate_models#Animation_types)
    ▪ <property> - Location in the property tree that defines how the layer will be oriented (http://wiki.flightgear.org/Property_Tree)
    ▪ <min>, <max>, <scale> - Set limits on animation

The gauges included in the Foxtrot Tango solution are detailed below as they may serve as a base model for instruments Moog may wish to include in their final product.
Altimeter

altimeter.xml

The altimeter gauge is used to measure the altitude of the aircraft in feet. Its layers include:

- Subscale - altimeter-subscale.png
  - /instrumentation/altimeter/setting-inhg
  - Rotates for precise altitude measurements
- Gauge face – altimeter-bg.png
  - No animation. Base image.
- Short needle – arrow.png
  - /instrumentation/altimeter/indicated-altitude-ft
  - Rotates to indicate altitude in thousands of feet
- Long needle – arrow.png
  - /instrumentation/altimeter/indicated-altitude-ft
  - Rotates to indicate altitude in hundreds of feet
- Button – button.png
  - No animation. Base image.
- Glare shield – glare-shield.rgb
  - No animation. Base image.

Airspeed Indicator

asi.xml

The airspeed indicator displays the airspeed of the aircraft in knots. Its layers include:

- Gauge face – asi-bg.png
  - No animation. Base image.
- Needle 1 – arrow.png
  - [TODO] property
  - Rotates to indicate speed in thousands of knots
- Needle 0 – arrow.png
  - [TODO] property
  - Rotates to indicate speed in hundreds of knots
- Glare shield – glare-shield.rgb
  - No animation. Base image.

Turn and Bank Indicator

turn.xml

The turn and bank indicator indicate the rate of a turn the aircraft is making. The gauge also shows the slip or skid of the turn. Its layers include:

- Gauge face – turn-bg.png
  - No animation. Base image.
- Tube-back – turn-tube-back.png
  - No animation. Base image.
- Ball – turn-ball.png
  - /instrumentation/slip-skid-ball/indicated-slip-skid
- Indicates the slip and/or skid of the turn the aircraft is making by rotating through the tube-back.

- Tube – turn-tube.png
  - No animation. Base image.

- Plane – turn-plane.png
  - /instrumentation/turn-indicator/indicated-turn-rate
  - Rotates to indicate which direction the plane is turning

- Glare shield – glare-shield.rgb
  - No animation. Base image.

**Vertical Speed Indicator**

vertical.xml

The vertical speed indicator or variometer shows the rate of climb or rate of descent of the aircraft is experiencing in feet per minute. Its layers include:

- Background – vertical-bg.png
  - Base image. No animation.

- Needle – arrow.png
  - /instrumentation/vertical-speed-indicator/indicated-speed-fpm
  - Rotates up/down to indicate rate of climb/descent

- Button – button.png
  - No animation. Base image.

- Glare shield – glare-shield.rgb
  - No animation. Base image.

**Artificial Horizon**

attitude-indicator.xml

The artificial horizon or attitude indicator shows the user the orientation of the aircraft relative to the earth. It shows the pitch and bank of the aircraft. Its layers include:

- Background – attitude-bg.png
  - /orientation/roll-deg
  - Rotates as the aircraft rotates

- Pitch – attitude-pitch.png
  - /orientation/pitch-deg
  - Rotates with the pitch of the aircraft. Also rotates with the background image

- Roll – attitude-roll.png
  - /orientation/roll-deg
  - Rotates with the roll of the aircraft.

- Plane – attitude-plane.png
  - /instrumentation/attitude-indicator/horizon-offset-deg
  - Moves with the orientation of the aircraft

- Foreground – attitude-fg.png
  - Base image. No animation.
**Directional Gyro**
gyro.xml
The directional gyro or heading indicator shows the pilot the aircraft’s heading. Its layers include:

- Compass rose – gyro-compass.png
  - /orientation/heading-deg
  - Rotates with the heading of the aircraft
- Fixed center – gyro-center.png
  - [TODO] property?
  - Base image. No animation.
- Glare shield – glare-shield.rgb
  - Base image. No animation

**Stall Warning**
stall-warning.xml
The stall warning lamp indicates when the engine has stalled in an aircraft. Its layers include:

- Stall light – stall-warning.png
  - /sim/alarms/stall-warning
  - Lights up when a stall warning has been triggered by the simulation
- Lens – stall-warning.png
  - Base image. No animation. Note that all components of this gauge are saved in the same image file.
## Class Roles & Responsibilities

<table>
<thead>
<tr>
<th>Class: ft_arinc_board_interface</th>
<th>Responsibilities: An abstract representation of an ARINC board.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collaborators</td>
<td>Uses: Used by: ft_board_manager</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class: ft_cei520a_board</th>
<th>Responsibilities: An abstract representation of the specific ARINC board, the CEI-520a. Holds data about how many channels the board has, etc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collaborators</td>
<td>Uses: Used by: ft_board_manager</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class: ft_board_manager</th>
<th>Responsibilities: Responsible for managing setup, teardown, and any read/write operations to one or more boards.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collaborators</td>
<td>Uses: ft_board_util Used by: ft_channel_monitor, ft_arinc_transmitter</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class: ft_channel_monitor</th>
<th>Responsibilities: A class that creates a thread to monitor a channel on an ARINC board. This class updates the ft_flightgear_properties class when new data is received.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collaborators</td>
<td>Uses: ft_board_manager, ft_flightgear_properties Used by:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class: ft_arinc_transmitter</th>
<th>Responsibilities: This class extends ft_gof_observer so that it can be notified when new data is ready to be transmitted via an ARINC card. When it receives that notify, it gets the update from the ft_flightgear_properties class and then encodes the data into a 32-bit ARINC word, and uses the ft_board_manager class to write the word to the card.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collaborators</td>
<td>Uses: ft_board_manager, ft_gof_observer Used by: ft_flightgear_properties</td>
</tr>
</tbody>
</table>
### Class: ft_arinc_dictionary

**Responsibilities:** This class keeps a hard coded list of ARINC labels and equipment ids. It essentially creates a mapping for defined variable constants to ARINC labels/equipment ids. Used by ft_flightgear_properties when decoding ARINC words and by ft_arinc_transmitter when encoding data into an ARINC word.

**Collaborators**

**Uses:**

- Used by: ft_flightgear_properties, ft_arinc_transmitter

### Class: ft_board_util

**Responsibilities:** This class uses the condor engineering libraries to access the board. The board manager is essentially using this utility class to perform board related operations (i.e setup board, start board, configure board, read/write)

**Collaborators**

**Uses:**

- condor engineering & bus tools DLLs
- Used by: ft_board_manager

### Class: ft_flightgear_properties

**Responsibilities:** An observable class (i.e extends the ft_gof_subject class) that stores the 12 variables needed by FlightGear. Can be updated in two ways: first, it can take in an ARINC word and a channel and then decode the data, or it can directly take in data and a variable constant. Calls its notify method whenever updated.

**Collaborators**

**Uses:**

- ft_gof_subject, ft_arinc_dictionary
- Used by: ft_channel_monitor, ft_arinc_transmitter

### Class: ft_gof_subject

**Responsibilities:** This is part of an implementation of the Gang of Four observer pattern. See documentation for “Gang of Four Observer Pattern”

**Collaborators**

**Uses:**

- ft_gof_observer
- Used by: ft_flightgear_properties

### Class: ft_gof_observer

**Responsibilities:** This is part of an implementation of the Gang of Four observer pattern. See documentation for “Gang of Four Observer Pattern”

**Collaborators**

**Uses:**

- N/A
- Used by: ft_gof_subject, ft_arinc_transmitter
Function Definitions

ft_board_manager

create_board

**SYNTAX DESCRIPTION:**

```c
int create_board(CEI_INT16 board_id, CEI_INT32 tx_channel, CEI_INT32 rx_channel, CEI_INT16 speed, CEI_INT16 wrap_setting, CEI_CHAR* api_version);
```

Creates a new board object with the given parameters.

**RETURN VALUE**

ARS_NORMAL Operation completed successfully.

ARS_INVBOARD Invalid board number

ARS_BRDNOTLOAD Board not initialized.

**ARGUMENTS**

- **Board_id** The id associated with the board being created.
- **Tx_channel** The transmit channel to be used
- **Rx_channel** The receive channel to be used
- **Speed** The speed setting to be used, either high or low (a Boolean switch)
- **Wrap_setting** The wrap setting, either internal or external (a Boolean switch)
- **Api_version** The api version, which can be obtained using the condor engineering function AR_VERSION

put_word

**SYNTAX DESCRIPTION**

```c
int put_word (CEI_INT16 board_id, CEI_INT32 tx_channel, CEI_INT32 word);
```

Puts a 32-bit word on the board at the specified transmit channel of the board associated
with the id passed in.

**RETURN VALUE**

ARS_NORMAL Operation completed successfully.
ARS_INVBOARD Invalid board number
ARS_BRDNOTLOAD Board not initialized.

**ARGUMENTS**

Board_id The id associated with the board being created.
Tx_channel The transmit channel to be used
Word The 32-bit word to be sent

---

**get_word**

**SYNTAX DESCRIPTION**

```c
int get_word ( CEI_INT16 board_id, CEI_INT32 rx_channel, CEI_INT32* word);
```

Gets a 32-bit word from the specified board on the specified rx_channel and stores it at the location specified by the pointer to a 32-bit word that is passed in via the “word” parameter.

**RETURN VALUE**

ARS_NORMAL Operation completed successfully.
ARS_INVBOARD Invalid board number
ARS_BRDNOTLOAD Board not initialized.

**ARGUMENTS**

Board_id The id associated with the board being created.
rx_channel The receive channel to be used
Word A pointer to a 32-bit word.
**put_word**

**SYNTAX DESCRIPTION**

```c
int start_board ( CEI_INT16 board_id);
```

After a board has been loaded and configured, this function is used to launch the board.

**RETURN VALUE**

- **ARS_NORMAL**  
  Operation completed successfully.
- **ARS_INVBOARD**  
  Invalid board number
- **ARS_BRDNOTLOAD**  
  Board not initialized.

**ARGUMENTS**

- **Board_id**  
  The id associated with the board being created.

---

**ft_board_util**

**setup_board**

**SYNTAX DESCRIPTION**

```c
int setup_board(ft_arinc_board_interface* p_board, CEI_INT32 tx_channel, CEI_INT32 rx_channel);
```

Takes in a pointer to a board object and a receive and a transmit channel and then configures the board. Note: this does not actually launch the board.

**RETURN VALUE**

- **ARS_NORMAL**  
  Operation completed successfully.
- **ARS_INVBOARD**  
  Invalid board number
- **ARS_BRDNOTLOAD**  
  Board not initialized.

**ARGUMENTS**

- **P_board**  
  A pointer to a board object, which holds all the necessary information about the board.
launch

**Syntax Description**

```c
int launch (ft_arinc_board_interface* p_board);
```

**Launches a board. Note: The board must be configured first using the setup_board function**

**Return Value**

ARS_NORMAL Operation completed successfully.

ARS_INVBOARD Invalid board number

ARS_BRDNOTLOAD Board not initialized.

**Arguments**

P_board A pointer to a board object, which holds all the necessary information about the board.

teardown_board

**Syntax Description**

```c
int teardown_board(ft_arinc_board_interface* p_board);
```

Takes in a pointer to a board object and then performs the necessary teardown operations to release the board and all memory associated with it.

**Return Value**

ARS_NORMAL Operation completed successfully.

ARS_INVBOARD Invalid board number

ARS_BRDNOTLOAD Board not initialized.
ARGUMENTS

P_board A pointer to a board object, which holds all the necessary information about the board.

ft_flightgear_properties

get_roll

**SYNTAX DESCRIPTION**

```c
double get_roll();
```

Returns the current value of roll angle.

**RETURN VALUE**

Roll the current value of roll

ARGUMENTS

N/A

get_pitch

**SYNTAX DESCRIPTION**

```c
double get_pitch();
```

Returns the current value of pitch angle.

**RETURN VALUE**

Pitch the current value of pitch

ARGUMENTS

N/A

get_yaw

**SYNTAX DESCRIPTION**

```c
double get_yaw();
```
Returns the current value of yaw angle.

**RETURN VALUE**

**Yaw**  the current value of yaw

**ARGUMENTS**

N/A

---

**get_latitude**

**SYNTAX DESCRIPTION**

double get_latitude();

Returns the current value of latitude.

**RETURN VALUE**

**Latitude**  the current value of latitude

**ARGUMENTS**

N/A

---

**get_longitude**

**SYNTAX DESCRIPTION**

double get_longitude();

Returns the current value of longitude.

**RETURN VALUE**

**LONGITUDE**  THE CURRENT VALUE OF LONGITUDE

**ARGUMENTS**

N/A
get_altitude

**Syntax Description**

double get_altitude();

Returns the current value of altitude.

**Return Value**

Altitude the current value of altitude

**Arguments**

N/A

get_roll_rate

**Syntax Description**

double get_roll_rate();

Returns the current value of roll rate.

**Return Value**

Roll rate the current value of roll rate

**Arguments**

N/A

get_pitch_rate

**Syntax Description**

double get_pitch_rate();

Returns the current value of pitch rate.

**Return Value**
Pitch rate       the current value of pitch rate

ARGUMENTS
N/A

get_yaw_rate

SYNTAX DESCRIPTION

double get_yaw_rate();

Returns the current value of yaw rate.

RETURN VALUE

Yaw Rate       the current value of yaw rate

ARGUMENTS
N/A

get_true_airspeed

SYNTAX DESCRIPTION

double get_true_airspeed();

Returns the current value of true airspeed.

RETURN VALUE

True airspeed       the current value of true airspeed

ARGUMENTS
N/A

update

SYNTAX DESCRIPTION
int update(CEI_INT16 channel, CEI_INT32 arinc_word);

Takes in a channel number and an arinc_word and uses the dictionary to decode the word and then updates the correct variable.

**RETURN VALUE**

ARS_NORMAL: Operation completed successfully.

**ARGUMENTS**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channel</td>
<td>The channel that the update came from</td>
</tr>
<tr>
<td>Arinc_word</td>
<td>The 32-bit word</td>
</tr>
</tbody>
</table>

**dspace_update**

**SYNTAX DESCRIPTION**

void dspace_update(int variable_constant, int data);

Takes in a variable constant and a data parameter and updates the associated variable. This is meant to be used when data is received from dSPACE, as the dSPACE data will not be encoded.

**RETURN VALUE**

N/A

**ARGUMENTS**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable_constant</td>
<td>The constant associated with the variable to be updated</td>
</tr>
<tr>
<td>data</td>
<td>The data</td>
</tr>
</tbody>
</table>

**ft_arinc_dictionary**

**get_equipment_id_by_channel**

**SYNTAX DESCRIPTION**

int get_equipment_id_by_channel(CEI_INT16 channel);

Takes in a channel and returns the equipment id that is associated with that channel.
Note: equipment id – channel associations are currently hardcoded in ft_arinc_definitions.

**RETURN VALUE**

`int` the equipment id associated with the given channel

**ARGUMENTS**

Channel The channel in question

---

**get_channel_by_channel_equipment_id**

**SYNTAX DESCRIPTION**

`int get_channel_by_channel_equipment_id(CEI_INT16 equipment_id);`

Takes in an equipment id and returns the channel that is associated with that equipment id. Note: equipment id – channel associations are currently hardcoded in ft_arinc_definitions.

**RETURN VALUE**

`int` the channel associated with the given equipment id

**ARGUMENTS**

Equipment id The equipment id in question

---

**get_full_arinc_label_by_variable**

**SYNTAX DESCRIPTION**

`int get_full_arinc_label_by_variable(CEI_INT16 variable_constant);`

Takes in a variable constant and returns the full ARINC label (equipment id + label) that is associated with that variable. Note: variable constants are currently hardcoded in ft_arinc_definitions.

**RETURN VALUE**

`int` the full label associated with the given variable constant
**Arguments**

Variable constant  The variable constant in question

---

**get_variable_constant_by_full_arinc_label**

**Syntax Description**

```c
int get_variable_constant_by_full_arinc_label (CEI_INT16 full_arinc_label);
```

Takes in a full ARINC label and returns the variable constant that is associated with that full ARINC label. Note: ARINC labels are currently hardcoded in ft_arinc_definitions.

**Return Value**

```c
int the variable constant associated with the given ARINC label
```

**Arguments**

Full_arinc_label  the ARINC label in question

---

**get_data_format_constant_by_variable_const**

**Syntax Description**

```c
int get_data_format_constant_by_variable_const (CEI_INT16 full_arinc_label);
```

Takes in a variable constant and returns the data format (i.e BCD, BNR, etc) that is associated with that variable. Note: ARINC labels are currently hardcoded in ft_arinc_definitions.

**Return Value**

```c
int the data format constant associated with the given variable
```
ARGUMENTS

Full_arinc_label the ARINC label in question

create_arinc_word

SYNTAX DESCRIPTION

CEI_INT32 create_arinc_word(CEI_INT16 arinc_label);

Takes in an ARINC label (does not include equipment id) and creates a 32-bit ARINC word. This overloaded version of create_arinc_word is used for variables that require two 32-bit ARINC words to get the required precision for the simulation. This function does not take in a data variable because it data is stored if the call to create_arinc_word(CEI_INT16 arinc_label, float data) detects that a second word is needed, then when this function is called, it calls create_arinc_word(CEI_INT16 arinc_label, float data) with the stored value as the data field.

RETURN VALUE

int the 32-bit ARINC word

ARGUMENTS

arinc_label the ARINC label
data The data

create_arinc_word

SYNTAX DESCRIPTION

CEI_INT32 create_arinc_word(CEI_INT16 arinc_label, float data);

Takes in an ARINC label (does not include equipment id) and a data variable and creates a 32-bit ARINC word.

RETURN VALUE
int the 32-bit ARINC word

**ARGUMENTS**

arinc_label the ARINC label
data The data

---

**requires_extended_precision**

**SYNTAX DESCRIPTION**

`bool requires_extended_precision (CEI_INT16 variable_const);`

Takes in a variable constant and returns a Boolean specifying whether or not the given variable requires extended precision. This is necessary because some variables in the ARINC specification overwrite the SDI and use the extra two bits to achieve a higher precision. (Example: Latitude and Longitude use 20 bits, when there are only 18 bits of data available normally).

**RETURN VALUE**

`bool` true if the variable requires extended precision, false otherwise

**ARGUMENTS**

Variable_const the variable constant associated with the variable in question

---

**requires_second_word**

**SYNTAX DESCRIPTION**

`bool requires_second_word(CEI_INT16 variable_const);`

Takes in a variable constant and returns a Boolean specifying whether or not the given variable requires a second 32-bit word for extended precision. This is necessary because some variables in the ARINC specification do not meet the precision required to smoothly run the simulation from dSPACE. (Example: Latitude, longitude, roll, pitch, yaw). Note: some variables require both extended precision (20 bits) and a second word. This is so that the first word still matches the ARINC specification. Note: extended precision words use custom labels and are BNR format with at most 18 data bits.
**RETURN VALUE**

`bool` true if the variable requires a second word, false otherwise

**ARGUMENTS**

`Variable_const` the variable constant associated with the variable in question

---

### is_second_word

**SYNTAX DESCRIPTION**

`bool is_second_word (CEI_INT16 variable_const);`

Takes in a variable constant and returns a Boolean specifying whether or not the given variable is the second word for a variable that requires a second word for extended precision.

**RETURN VALUE**

`bool` true if the variable is a secondary/extended precision word, false otherwise

**ARGUMENTS**

`Variable_const` the variable constant associated with the variable in question

---

### convert_to_double

**SYNTAX DESCRIPTION**

`double convert_to_double (CEI_INT32 label, int data);`

Takes in a full label (lable and eq id) and the data as and int (assumed to be the 18 data bits shift so the LSB is bit 0) and returns the data converted to a double. This is necessary because of the data formats ARINC uses. This function is used by FlightGear properties when it receives an update.

**RETURN VALUE**

`double` The data converted to a double
ARGUMENTS

label the label

data the data bits from an ARINC word

print_flightgear_variable_constants

SYNTAX DESCRIPTION

void print_flightgear_variable_constants();

Prints a list of the FlightGear variable constants.

RETURN VALUE

N/A

ARGUMENTS

N/A

ft_channel_monitor

begin_monitoring_thread

SYNTAX DESCRIPTION

int begin_monitoring_thread();

This method starts the thread that monitors the ARINC channel that this object has been configured to monitor.

RETURN VALUE

N/A

ARGUMENTS

N/A

request_stop

SYNTAX DESCRIPTION
int request_stop();

Turns the flag “stop_requested” to true, so that the thread will stop the next time it checks its loop condition. This way, the thread is not stopped forcefully.

**RETURN VALUE**

N/A

**ARGUMENTS**

N/A

---

**s_stop_requested**

**SYNTAX DESCRIPTION**

bool is_stop_requested();

Used by the thread to determine when it should stop.

**RETURN VALUE**

Bool The current value of stop_requested

**ARGUMENTS**

N/A

---

**get_board_manager_ptr**

**SYNTAX DESCRIPTION**

ft_board_manager* get_board_manager_ptr();

Accessor for the ptr to the board_manager.

**RETURN VALUE**

ft_board_manager* A pointer to the ft_board_manager

**ARGUMENTS**
get_flightgear_properties_ptr

**Syntax Description**

```c
ft_flightgear_properties * get_flightgear_properties_ptr();
```

Accessor for the ptr to the ft_flightgear_properties.

**Return Value**

`ft_flightgear_properties *` A pointer to the ft_flightgear_properties

**Arguments**

N/A

get_board_id

**Syntax Description**

```c
CEI_INT16 get_board_id();
```

Accessor for the board_id.

**Return Value**

`CEI_INT16` Board id.

**Arguments**

N/A

get_channel

**Syntax Description**

```c
CEI_INT16 get_channel();
```

Accessor for the channel
RETURN VALUE

CEI_INT16 the channel

ARGUMENTS

N/A

get_arinc_word_ptr

SYNTAX DESCRIPTION

CEI_INT32* get_arinc_word_ptr();

Accessor for the pointer to the ARINC word variable.

RETURN VALUE

CEI_INT32* A POINT TO THE ARINC WORD VARIABLE

ARGUMENTS

N/A

ft_arinc_transmitter

update

SYNTAX DESCRIPTION

void update(ft_gof_subject* changed_subject);

This method is an implementation of the method update from the superclass ft_gof_observer. In this case, update gets the variable that was most recently changed in ft_flightgear_properties and then creates an ARINC word based off the variable and the value and puts it on the transmit channel being used.

RETURN VALUE

N/A

ARGUMENTS

ft_gof_subject* The subject that called the update method.
*For information on ft_gof_subject and ft_gof_observer, see documentation for “gang of four” observer.
Class Diagram

Adding/Removing Variables

To add a variable, you must first create a new variable constant for the variable. This is done by adding a constant in the ft_arinc_definitions.h file. The naming convention is FT_variable_name. Then you must look up the ARINC label and create a constant for that as well in the ft_arinc_definitions.h file. Next, if the variable is BNR format, then you must create constants for the padding and the “BNR constant”. These can be found in the ARINC specification. The BNR constant is the maximum value that is found in the ARINC specification (i.e for Roll 180, for 180 degrees). The next step is to update the hardcoded constructor for the dictionary. This is in ft_arinc_dictionary.cpp, inside its constructor. It is currently hardcoded, but could be modified in the future to use an XML configuration file.

The next step is updating switch statements in the following files:

Ft_flightgear_properties.cpp
Ft_arinc_transmitter.cpp
Xml_parser.cpp
Each contains switch statements that are dependent on variable constant, so cases need to be added for each new variable.

Next, a field and accessor method will be added to the ft_flightgear_properties.cpp class for the new variable. For example if you were creating a variable for roll, you would create a private double called roll and then a public get_roll() function.

Next, add print statements to the print_variables function in ft_arinc_dictionary.cpp.

If you are adding a variable that requires extended precision, make sure its padding is set to -2, and update the requires_extended_precision function in ft_arinc_dictionary.cpp to include the new variable.

If you are adding a variable that needs two ARINC words of precision, then you must create a second variable const called FT_my_variable_EXTEND. Then find a suitable custom label and add that to the definitions file as well. Again you need padding and BNR constant info. Next update the hard coded constructor as you would with the normal variable. Then update the requires_second_word and is_second_word functions appropriately (they contain switch statements).

Finally, the switch statements in FlightGear properties should not do anything for the first word if a variable has two words. For example if you create FT_ROLL and FT_ROLL_EXTEND, then flightgear_properties should not do anything when it receives FT_ROLL, the update should occur when it receives FT_ROLL_EXTEND.

The last thing to do is go to the FT_NUM_VARS constant and increment it by 1. Regardless of whether you added a variable that takes one or two words, only in
Appendix C – User Setup

Installing
The installation can be done using either Setup1.msi or Setup.exe. This should be run one time and will install all DLLs that are needed to run our programs.

Executables
We have included several executables to allow you to run our code. First, there is a wrap simulator called ft_arinc_wrap_sim.exe that allows the user to test the software on one computer using one arinc card. The program is a command line interface and allows you to first configure the board and then send individual words of data to the card and read them back.

Next, there are send and receive simulators called ft_arinc_send_simulator.exe and ft_arinc_receive_simulator.exe. Similarly, the user can setup the board and then send individual packets, but this is meant for testing across two machines with an arinc cable.

Next, there is the ft_dspace_file_parse.exe. This program allows the user to setup the board and then it parses a test file and sends the entire file across the arinc hardware.

Finally there is a receive and send executable called FT_XML_send.exe. This program allows the user to setup the arinc board and then sends data to flightgear as it receives it. Finally, there is an executable called ft_dspace_file_parse.exe that reads a text file, parses it and sends the data across the ARINC hardware. This can be used to simulate dSPACE. Eventually this will need to be an actual interface to the dSPACE simulation. To do this, a new executable can be made that opens a network socket, receives data from dSPACE on that socket, and then sends the data to the hardware (this could be modeled after the ft_dspace_file_parse.exe code as it would function almost exactly the same way).

In all the executables, the user will be prompted for a board id (default 0), transmit and receive channels (which should be set to 1), and whether the user wishes to do an internal or external simulation (should be internal if it is wrap sim, external otherwise), and finally the bit rate (either high or low) which should be set to high.

In send sim and wrap sim, the user needs to select a variable and then a value for the variable. The user can also hit h for help.

Configuration
The directory Aircraft/FlyingTiger should be copied to C:\Program Files\FlightGear\data\Aircraft (path may vary). This directory contains all the configuration files needed to properly setup the displays in FlightGear. For more accurate rendering, the file ft-config.xml should be modified to the correct resolution and distance from the screen. Within this file, the variables PropertyList/sim/rendering/camera-group/camera/window/width and height (marked with “TODO”) should be changed to match the resolution of the environment display in pixels. Likewise, the PropertyList/sim/rendering/camera-group/camera/view/x, y, and z should be changed such that they define the distance between the center of mass and the center point of the screen in meters. Comments next to these lines indicate the directions.
Additionally, The file Protocol/ft-test.xml should be copied to C:\Program Files\FlightGear\data\Protocol.

The first time the FlightGear Wizard is run, various paths need to be set. Note that because the install path may vary these paths may not be exactly the same.

- Executable = C:\Program Files\FlightGear\bin\Win32\fgfs.exe
- FG_ROOT = C:\Program Files\FlightGear\data
- FG_AIRCRAFT = C:\Program Files\FlightGear\data\Aircraft
- FG_SCENERY = C:\Program Files\FlightGear\data\Scenery

On the next screen, select the “Moog flight simulator” aircraft model. Next, select the desired map. If this map does not match the coordinates used by the dSPACE flight dynamics model, the aircraft will “jump” to the coordinates of the dSPACE model. On the last screen, select “Advanced”, this will popup a new window. Under the General tab, set “Config” to “C:\Program Files\FlightGear\data\Aircraft\FlyingTiger\ft-config.xml”. In the Flight Model tab, set FDM to “external”. On the Input/Output tab, add a new entry with the following settings:

- Protocol = generic
- Medium = socket
- Direction = in
- Hz = 64
- Port = 5050
- Select UDP
- Generic = ft-test

Various other settings can be configured from here as well, including rendering options under the Features tab. Enabling TerraSync will allow FlightGear to fetch real-world terrain data when the simulation flies off the selected map.