NASA/TM-2008-214639

A Software Framework for Aircraft Simulation

Brian P. Curlett
NASA Dryden Flight Research Center
Edwards, California

October 2008

NASA/TM-2008-214639

A Software Framework for Aircraft Simulation

Brian P. Curlett
NASA Dryden Flight Research Center
Edwards, California

National Aeronautics and
Space Administration

Dryden Flight Research Center
Edwards, California 93523-0273

October 2008

Available from:

NASA Center for AeroSpace Information
7115 Standard Drive
Hanover, MD 21076-1320
(301) 621-0390

ABSTRACT

The National Aeronautics and Space Administration Dryden Flight Research Center has a long history in developing simulations of experimental fixed-wing aircraft from gliders to suborbital vehicles on platforms ranging from desktop simulators to pilot-in-the-loop / aircraft-in-the-loop simulators. Regardless of the aircraft or simulator hardware, much of the software framework is common to all NASA Dryden simulators. Some of this software has withstood the test of time, but in recent years the push toward high-fidelity user-friendly simulations has resulted in some significant changes. This report presents an overview of the current NASA Dryden simulation software framework and capabilities with an emphasis on the new features that have permitted NASA to develop more capable simulations while maintaining the same staffing levels.

NOMENCLATURE

3D	three-dimensional
ARINC	Aeronautical Radio, Incorporated
CIU	cockpit interface unit
CVT	current value table
DTH	Dryden time history (data file formats)
GUI	graphical user interface
HIU	hardware interface unit
HUD	head-up display
HTML	Hypertext Markup Language
I/O	input/output
IP	Internet Protocol
IRIG	Inter Range Instrumentation Group
JNI™	Java™ Native Interface
MCC	Mission Control Center
NASA	National Aeronautics and Space Administration
PC	personal computer
PCI	Peripheral Component Interconnect
RAIF	Research Aircraft Integration Facility
RES	real-time Ethernet server
RT3D	Real-time 3D (graphics program)
SES	simulation electric stick
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

USGS	United States Geological Survey
VME	Versa Module Eurocard (computer standard)

1. INTRODUCTION

For fifty years, simulators, first analog and then digital, have played a critical role in flight research. Gene Waltman gives an excellent account of early flight simulation in *Black Magic and Gremlins: Analog Flight Simulations at NASA's Flight Research Center* (ref. 1). In those early years, pilots and engineers were skeptical of the usefulness of simulators, which were then expensive devices the results from which could change with the temperature of the room in which they were housed. But in the 1960s, when the National Aeronautics and Space Administration (NASA) Dryden Flight Research Center (Edwards, California) was heavily involved in hypersonic and lifting body research, it was not possible for pilots to gain experience behind the control stick of these one-of-a-kind experimental aircraft (a fact that is still true today). Pilots that spent more hours practicing in simulators were better able to meet test objectives than those with less training. This, combined with the success of the early simulators in predicting some unexpected handling characteristics, helped establish simulation as a vital part of most flight research programs. The advances in digital computers during the 1970s and 1980s enabled more realistic modeling of the aircraft, and pilots not only put more trust in simulation results, but demanded that simulators be provided for most flight research projects (ref. 2).

As the role of the simulator in flight research continued to grow, NASA Dryden progressed from one or two reconfigurable simulators to a building containing dedicated simulation laboratories for each flight research project. The Research Aircraft Integration Facility (RAIF) at NASA Dryden was designed to support simulation, vehicle system integration, and full vehicle integrated testing (ref. 3). The RAIF currently houses six aircraft test bays and up to eleven closely located simulation laboratories. By the time this facility opened in 1992, NASA Dryden already had a well-established simulation software framework (most of which was written in the Fortran computer programming language) capable of batch mode, real-time pilot-in-the-loop, hardware-in-the-loop, and aircraft-in-the-loop operation (ref. 4). Dedicated simulation laboratories, designed for operation by a single user, meant that simulators could be used for more of the day-to-day flight research activities. Currently the simulators located in the RAIF are used for engineering analysis more than for pilot training. Typical simulation tasks include evaluation of new vehicle concepts, control law development and validation, flight safety analysis, mission planning, flight envelope expansion, and postflight data analysis.

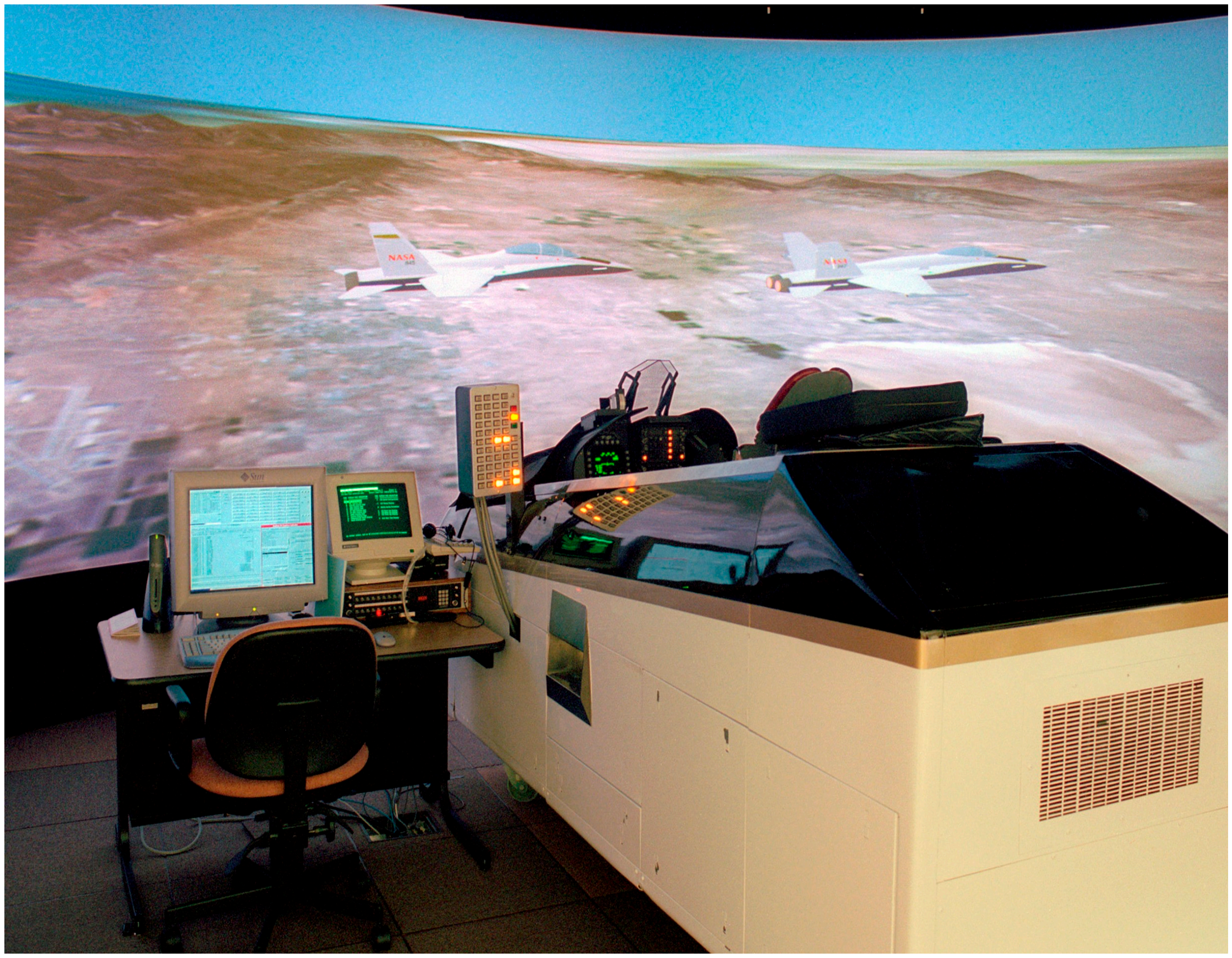
The disadvantage of having dedicated resources for each flight research project was that the simulation software began to diverge as each flight research project made the changes that were needed to meet their research objectives. In recent years, an effort was made to reestablish a common software structure for all NASA Dryden simulations; this goal was accomplished by collecting the best practices from each flight research project team and developing generic code that can be used for all fixed-wing aircraft simulations. All current simulations at NASA Dryden are based on this common software framework, called the “Core,” which is the main topic of this report. The Core is used for aircraft simulations ranging from gliders to suborbital vehicles and runs on platforms ranging from laptop computers to pilot-in-the-loop / aircraft-in-the-loop simulators.

The Core is composed of standard models, mathematical routines, a user interface, hardware interfaces, timing routines, data recording and data input subsystems, external application interfaces, and other sharable modules. The Core is predominately written in the C++ computer programming language but supports legacy Fortran models. This report first provides a description of the hardware that is used in a typical simulator at NASA Dryden, and then provides a more detailed overview of the Core. Sections 4 and 5 discuss the graphical user interface (GUI) and three-dimensional (3D) out-the-window graphics, respectively. The remainder of the report (sections 6 through 17) describes each of the main features of the Core, including real-time performance, the data dictionary, scripting, the Calc language, initial conditions, the simulation log, file input/output, automated testing, Mission Control Center interface, real-time Ethernet server, external application interfaces, and data bus tools. Section 18 presents concluding remarks and Appendix A provides a listing of the display pages which are first introduced in section 4.

2. SIMULATOR HARDWARE

Simulator hardware needs vary greatly from one flight research project to another. The NASA Dryden simulators operate on platforms ranging from a standalone laptop computer to pilot-in-the-loop / aircraft-in-the-loop laboratories containing dozens of multiprocessor computers. This section will describe a hardware configuration that is typical of one of the more complete simulation laboratories at NASA Dryden. It should be noted that the same simulation software is also used in much simpler configurations.

Simulator cockpits are fixed-based and include most of the primary flight controls and instruments that are found in the aircraft cockpit. Since the simulators are used predominately for ground testing and engineering analysis (and not for pilot training) none of the simulator cockpits represent a fully functional aircraft. Figure 1 shows a simulator cockpit for an F-18 (McDonnell Douglas, now the Boeing Company, Chicago, Illinois) airplane. The in-house-developed simulation electric stick (SES) uses high-torque electric servo motors to provide realistic forces and damping for center control stick and rudder pedal applications. The cockpit interface unit (CIU) provides all of the analog and discrete input/output (I/O) to the simulator cockpit instruments and controls. Both the CIU and the SES use a combination of off-the-shelf Versa Module Eurocard (VME) and custom hardware. The software for both systems is written in the C computer programming language and operates under the VxWorks® (Wind River Systems, Inc., Alameda, California) real-time computer operating system on single-board computers. The CIU and SES communicate with the main simulation computer by way of a high-speed, low-latency, reflective memory. Figure 2 illustrates the basic interconnections between the hardware components.



EC04-0288-2

Figure 1. F-18 fixed-base cockpit with simulation operator’s console and wide screen out-the-window visuals.

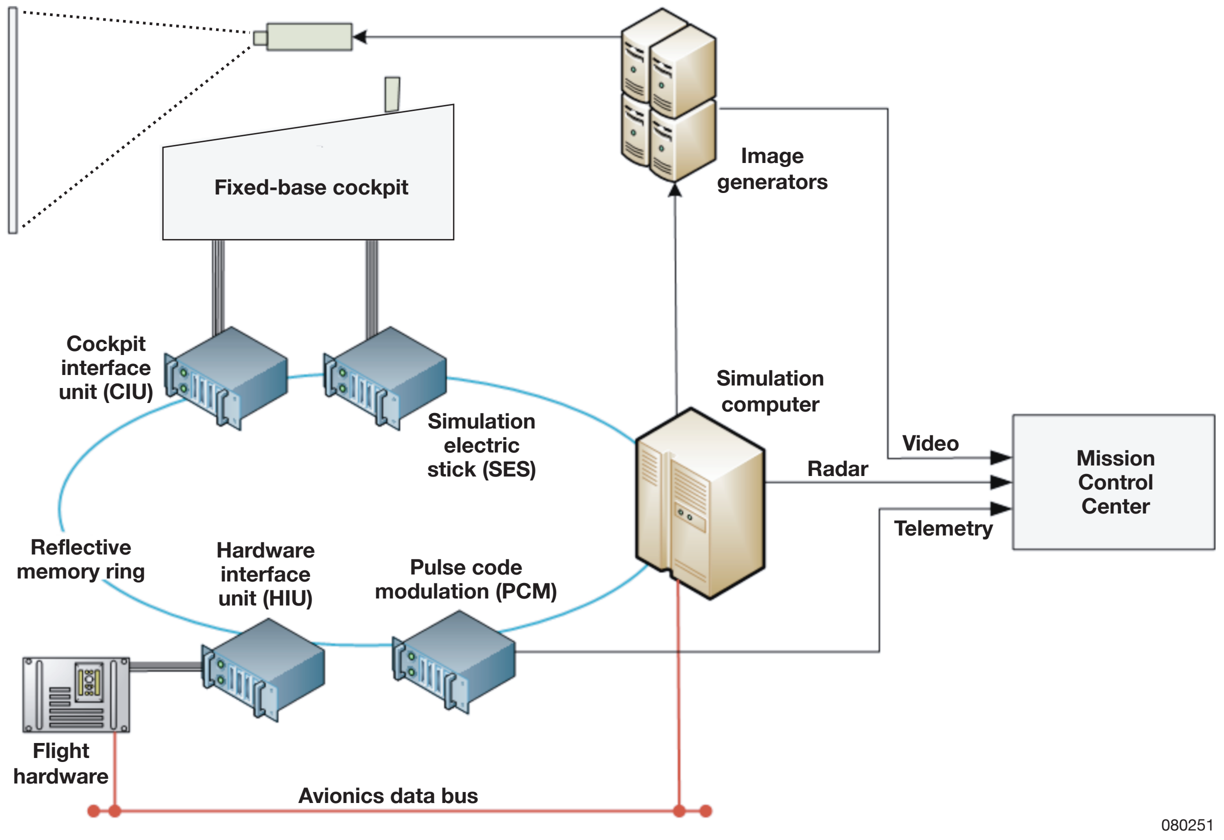


Figure 2. Basic interconnections of simulator hardware components.

Multifunction displays in the simulator cockpit are either actual aircraft hardware or simulated using touchscreen displays. In either case, a flight hardware mission computer is connected into the simulation to provide data processing for the digital display units. A mission computer normally only requires power, cooling air, and a data bus interface to the simulation computer and display units. Other flight hardware, such as flight control computer hardware, requires extensive analog and digital interfaces that are provided by a hardware interface unit (HIU) that is very similar to a CIU but contains additional circuits to protect these often one-of-a-kind flight hardware components.

The main simulation computer is usually a symmetric multiprocessor running a variation of the Unix computer operating system. A server-class computer with four to eight processors and eight or more expansion slots is normally required for a pilot-in-the-loop / hardware-in-the-loop simulator. Currently, most laboratories utilize the Sun Microsystems, Incorporated (Santa Clara, California) Sun Fire™ V890 server running the Solaris™ 10 operating system. Although the vast majority of I/O is handled by the CIU, SES, and HIU, the simulation computer requires several interface boards such as MIL-STD-1553 and ARINC-429 (Aeronautical Radio, Incorporated, Annapolis, Maryland) bus interfaces, serial interfaces, an Inter Range Instrumentation Group (IRIG) IRIG-B time code reader, telemetry encoders, and the reflective memory interface to the CIU, SES, and HIU. At times, Peripheral Component Interconnect (PCI) PCI-to-PCI or PCI-to-VME expansion buses have been used to accommodate these and other interface boards. X terminals or personal computers (PCs) running an X server application are used to provide additional consoles to the main simulation computer. A second, lower-end simulation computer without hardware interfaces is normally provided for software development.

The visual system is typically a front-projection system with a large wall-mounted screen; however, liquid crystal display (LCD) or plasma screens are used in some laboratories. In recent years, low-cost PC hardware has been used as image generators. Image generators are interfaced to the simulation computer by way of an Ethernet connection. Graphics software is described in section 5 below.

Most of the NASA Dryden simulators have a mechanical or a touchscreen control panel, seen in figure 1, mounted on the side of the simulator cockpit. The simulator pilot can use this control panel to start, pause, and reset the simulation; slew initial conditions; and control other simulation options. This permits one-man operation of the simulation from inside the simulator cockpit, in many cases eliminating the need for a simulator operator.

Displays from the Mission Control Center (MCC), such as a ground-track map or digital strip charts, are also used in the simulation laboratories. These PC applications are either driven directly by the simulation using an Ethernet connection, or indirectly by encoding a telemetry stream into the simulation and decoding the stream using a telemetry processor similar to the units used in the MCC. The software to accomplish the latter is quite elaborate and is described in section 14 below.

3. CORE SOFTWARE

All current simulations at NASA Dryden are based on a common software framework called the Core. The Core consists of standard simulation models, a user interface, hardware interfaces, timing routines, data recording and data input subsystems, external application interfaces, and other sharable modules. The Core as a standard framework allows simulation developers to focus on modeling what is unique to each aircraft, such as aerodynamics, engine performance, control system, mass properties, gear dynamics, other effectors, and avionics.

Currently, the complete source code tree for each simulator is revision-controlled separately, permitting each flight research project team to control the timing of their software updates, and to customize the Core as necessary. If a project team makes changes that are useful to other project teams, the changes can be incorporated into the master copy of the Core.

The master copy of the Core is revision-controlled as a complete simulation with generic models to facilitate testing. To ensure that the Core only contains well-tested code that represents best practices, a configuration control board composed of all the members of the simulation branch oversees all changes to the master copy of the Core. Flight-research-project-specific simulation software is controlled by the pertinent flight research project team; the level of oversight depending upon the development phase of the software.

The focus of this report is the framework portion of the Core, but the Core also includes some common models such as the equations of motion, atmospheric properties, terrain, and wind models. The six-degrees-of-freedom rotating oblate Earth equations of motion are derived in ref. 5 (NASA internal memorandum). These differential equations are integrated using either a modified second-order Runga Kutta (ref. 6) (NASA internal memorandum) or a fourth-order Runga Kutta scheme; giving the flight research project team a choice between speed and accuracy. The 1976 standard atmosphere is used. A constant wind profile as a function of altitude can be selected. Winds can be perturbed with a one minus cosine shaped gust in the aircraft body axis as a function of time, or as random turbulence with a selectable severity range. The terrain height field is interpolated from a United States Geological Survey (USGS) digital elevation model with 3 arc-second spacing. Models for aircraft aerodynamics and engine performance vary from one flight research project to another, but are usually based on tabulated empirical data. Custom code for highly efficient table lookup can be automatically generated in C or Fortran (ref. 4), or the generic table lookup functions from the Core mathematics library can be used. Standardized models of actuator and gear dynamics are currently being developed, along with a generic control system as described in reference 7.

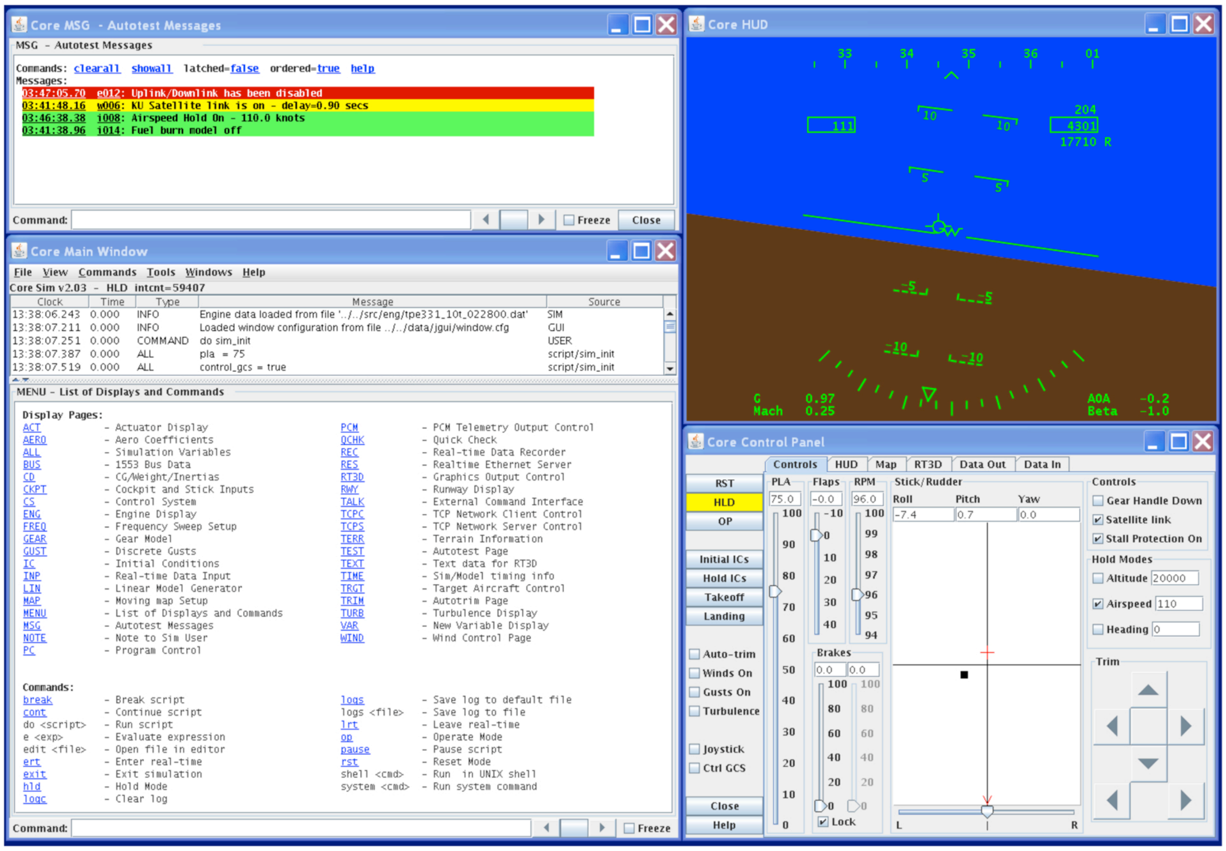
Most of the Core is now object-oriented C++ code, but some legacy Fortran remains. Use of C code has been phased out in favor of procedural C++ code, because the C++ compiler more accurately detects programming errors and results in a slightly more efficient and easier-to-read code. The C++ code and the Fortran code is compiled together and the data are shared for the most part by overlaying the C data structures with Fortran common blocks. Both the Fortran structures and the C structures are automatically generated from data dictionaries (see section 7 below) to reduce the coding effort and to avoid hard-to-detect memory alignment problems. Although this hybrid of object-oriented and procedural code might not be ideal, a high priority has not been to replace code that is perceived as tried and true. The user interface is written in the Java™ (Sun Microsystems, Inc., Santa Clara, California) computer programming language. Control system models are sometimes provided in the Ada computer programming language, but these models are compiled into separate executables that exchange data with the simulation using shared memory and semaphores.

4. USER INTERFACE

The Core provides a GUI written in the Java™ programming language. Java™ was selected for its portability, ease of use, and extensive graphics libraries. The Java™ code interfaces with the simulation using the Java™ Native Interface (JNI™) (Sun Microsystems, Inc., Santa Clara, California). This mechanism permits Java™ to perform C++ function calls to access data in the simulation's symbol table, command queue, and other C++ objects.

In recent years, many graphical controls have been added to the GUI. The Core Main Window, shown in the lower left of figure 3, contains the main menu, the status bar, the simulation log, the main display area, and a command prompt. The log keeps a time-stamped record of all user inputs and any responses from the simulation. The display area shows continuously updating text for the currently selected model. Hypertext Markup Language (HTML) encoding can be used in the display area to provide rich text formatting, color, and hyperlink commands. At the command prompt the user enters commands into the simulation. Secondary windows, seen in the top left of figure 3, also include a display area and command prompt but do not include the main menu or simulation log. The user can open multiple secondary displays to view data from multiple models simultaneously.

The Core Control Panel window, seen in the lower right of figure 3, provides graphical controls to change initial conditions and to fly the simulation using a mouse computer pointing device. The Core Control Panel window also provides controls on separate tabs for common tasks such as controlling the visual system and data recorder. The Core HUD window, seen in the upper right of figure 3, is a simple graphic based on the F-18 head-up display (HUD) that allows the simulation to be operated when the full 3D graphics are not available.



```
MSG - Autotest Messages

Commands: clearall  showall  latched=false  ordered=true  help
Messages:
 03:47:05.70  e012: Uplink/Downlink has been disabled
 03:41:48.16  w006: KU Satellite link is on - delay=0.90 secs
 03:46:38.38  i008: Airspeed Hold On - 110.0 knots
 03:41:38.96  i014: Fuel burn model off
```

```
Core Sim v2.03 -  HLD  intcnt=59407
   Clock        Time   Type      Message                                                          Source
13:38:06.243  0.000  INFO      Engine data loaded from file '../../src/eng/tpe331_10t_022800.dat'  SIM
13:38:07.211  0.000  INFO      Loaded window configuration from file ../../data/jgui/window.cfg    GUI
13:38:07.251  0.000  COMMAND   do sim_init                                                         USER
13:38:07.387  0.000  ALL       pla  = 75                                                           script/sim_init
13:38:07.519  0.000  ALL       control_gcs = true                                                  script/sim_init
```

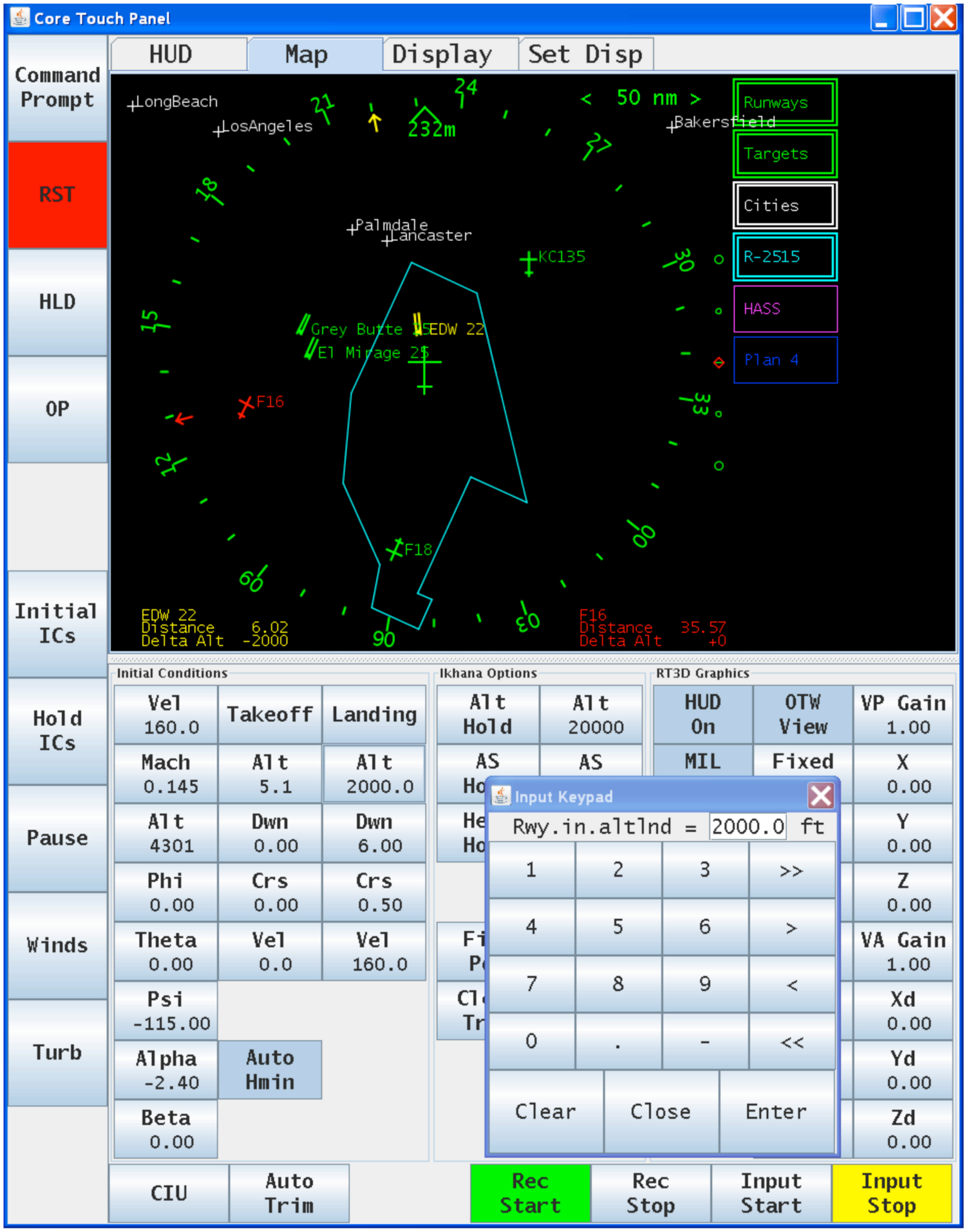
```
MENU - List of Displays and Commands

Display Pages:
ACT           - Actuator Display                 PCM           - PCM Telemetry Output Control
AERO          - Aero Coefficients                QCHK          - Quick Check
ALL           - Simulation Variables             REC           - Real-time Data Recorder
BUS           - 1553 Bus Data                    RES           - Realtime Ethernet Server
CD            - CG/Weight/Inertias               RT3D          - Graphics Output Control
CKPT          - Cockpit and Stick Inputs         RWY           - Runway Display
CS            - Control System                   TALK          - External Command Interface
ENG           - Engine Display                   TCPC          - TCP Network Client Control
FREQ          - Frequency Sweep Setup            TCPS          - TCP Network Server Control
GEAR          - Gear Model                       TERR          - Terrain Information
GUST          - Discrete Gusts                   TEST          - Autotest Page
IC            - Initial Conditions               TEXT          - Text data for RT3D
INP           - Real-time Data Input             TIME          - Sim/Model timing info
LIN           - Linear Model Generator           TRGT          - Target Aircraft Control
MAP           - Moving map Setup                 TRIM          - Autotrim Page
MENU          - List of Displays and Commands    TURB          - Turbulence Display
MSG           - Autotest Messages                VAR           - New Variable Display
NOTE          - Note to Sim User                 WIND          - Wind Control Page
PC            - Program Control


Commands:
break         - Break script                     logs          - Save log to default file
cont          - Continue script                  logs <file>   - Save log to file
do <script>   - Run script                       lrt           - Leave real-time
e <exp>       - Evaluate expression              op            - Operate Mode
edit <file>   - Open file in editor              pause         - Pause script
ert           - Enter real-time                  rst           - Reset Mode
exit          - Exit simulation                  shell <cmd>   - Run  in UNIX shell
hld           - Hold Mode                        system <cmd>  - Run system command
logc          - Clear log
```

```
PLA 75.0   Flaps -0.0   RPM 96.0
Stick/Rudder   Roll -7.4   Pitch 0.7   Yaw 0.0
Brakes 0.0 0.0   [✓] Lock
Controls
[ ] Gear Handle Down
[✓] Satellite link
[✓] Stall Protection On
Hold Modes
[ ] Altitude 20000
[✓] Airspeed 110
[ ] Heading 0
```

080252

Figure 3. Simulation GUI showing (clockwise from lower left) the Main Window (including the main menu, simulation log, display page, and command prompt), the Message Stack, the HUD Display, and the Cockpit Control Panel.

For a pilot-in-the-loop simulation the GUI is normally displayed on a console located beside the cockpit. The operator's console can also include a second touchscreen monitor mounted on the side of the cockpit permitting a single person to operate the simulation while seated in the cockpit. The GUI provides a special control panel, called the Core Touch Panel and shown in figure 4, with oversized buttons suitable for use on a touchscreen display. Touching an initial condition button will result in a number pad pop-up window to be used for data entry. The user can also initiate a full alphanumeric keyboard graphic pop-up to be used for command entry. The top display area of the touchscreen window can be set to the HUD graphic, Map/Radar graphic (shown in figure 4), or any simulation display page. The Map/Radar graphic shows the location of runways and other aircraft relative to the simulated airplane and can also display flight plans, city locations, and restricted airspace.



EDW 22
Distance 6.02
Delta Alt -2000

F16
Distance 35.57
Delta Alt +0

Vel 160.0	Takeoff	Landing
Mach 0.145	Alt 5.1	Alt 2000.0
Alt 4301	Dwn 0.00	Dwn 6.00
Phi 0.00	Crs 0.00	Crs 0.50
Theta 0.00	Vel 0.0	Vel 160.0
Psi -115.00		
Alpha -2.40	Auto Hmin	
Beta 0.00		

Alt Hold	Alt 20000

HUD On	OTW View	VP Gain 1.00
MIL	Fixed	X 0.00
		Y 0.00
		Z 0.00
		VA Gain 1.00
		Xd 0.00
		Yd 0.00
		Zd 0.00

Rwy.in.altlnd = 2000.0 ft

080253

Figure 4. Touchscreen control panel with Map/Radar display and popup number pad.

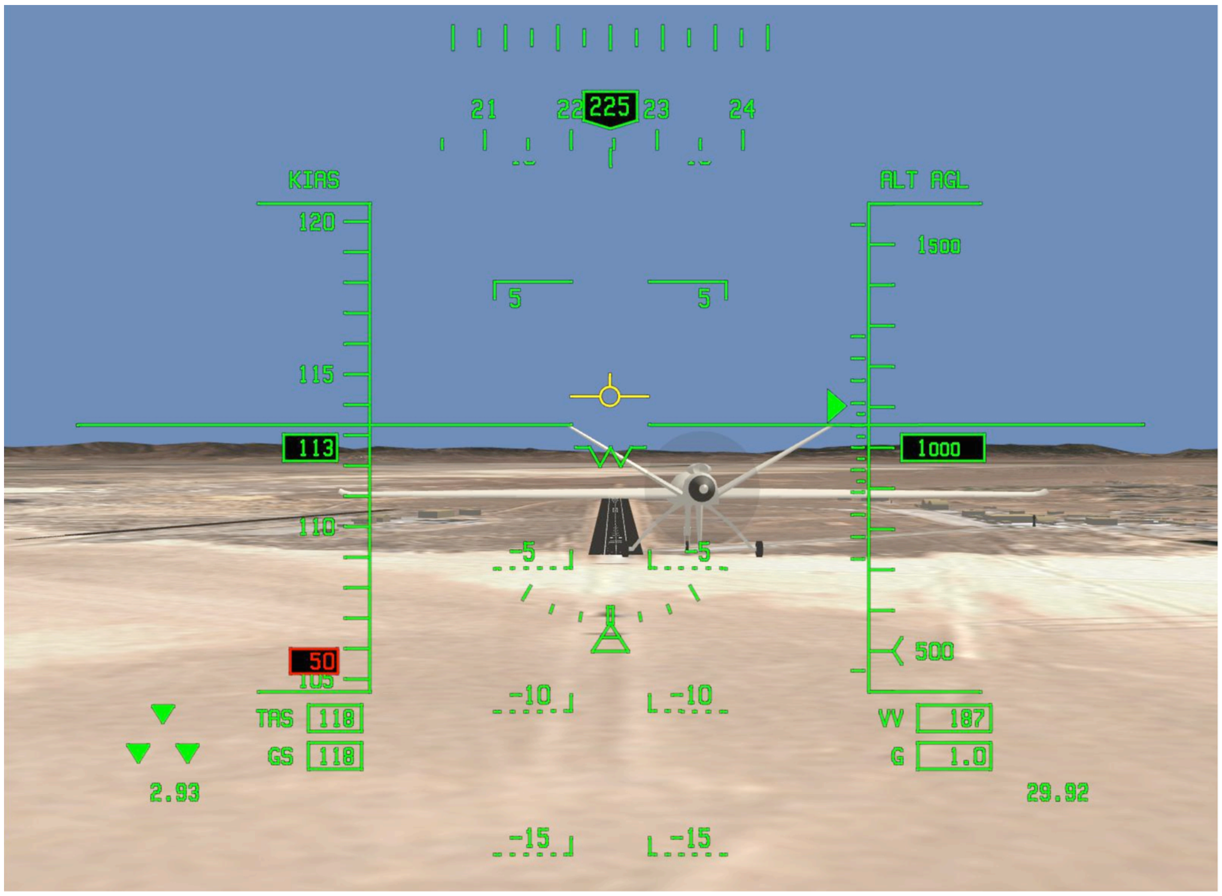
The GUI contains additional windows that are not shown in the figures included with this report. The data dictionary browser and the variable list windows permit the user to explore, search, display, and modify all data in the simulation's symbol table. The MIL-STD-1553 bus monitor permits the display and recording of all bus traffic including both raw and engineering unit data. The Editor window provides a simple text editor for scripts and plain text data files. Finally, the Help window displays all of the HTML documentation for the simulation without the need to bring up a full web browser. The screen layout for all the GUI windows, including those on the touchscreen monitor, can be changed by the user and saved to a configuration file.

Although the GUI now has many menus, control panels, and other graphical objects, the heart of the user interface remains text-based display pages and a command prompt. There is normally a display page for each aircraft model (aerodynamics, engine, mass properties, et cetera) and for each framework module (data recorder, telemetry encoder, cockpit interface, et cetera). A display page has two purposes: to continuously refresh the text shown on the screen and to process commands relevant to the currently displayed text. A typical display page will show all of the inputs and outputs, and the more important internal parameters, of a model. The display page may also display a list of commands that the model can process for quick reference. These commands are predominately for setting options on the model or module. In addition to all of the model displays, there are a few general displays that provide summary data or permit the user to select a customized list of parameters to display and modify. A list of common display pages is provided in Appendix A.

The text update logic and command processing code for a given display page are included as part of the model to keep the simulation modular. Models can be added or removed from the simulation without changing the Java™ GUI code or the command processor, and because display pages are text-based they can be programmed in the same language as the model. Also, to help maintain modularity, the documentation for each display page is written as a separate HTML file and revision-controlled with the model.

5. THREE–DIMENSIONAL VISUALS

RT3D (Real-time 3D) is an in-house-developed software package that renders the out-the-window visuals for the simulator screen. A screen capture of an out-the-window visual is shown in figure 5. RT3D can display the Earth and the aircraft from several points of view such as the pilot’s view, the chase pilot’s view, a fixed camera view, or a long-range optics view. Terrain, runways, and buildings are modeled in the areas around several airfields, including Dugway, Edwards, El Mirage, Fairchild, Grey Butte, and Mountain Home. Beyond these areas the Earth is modeled as a smooth oblate surface with a coarse texture map. Aircraft models include articulating control surfaces, landing gear, jet plumes, and any other required details. Smoke generators are supported on vehicles so equipped and time-history traces (similar to a contrail) are supported on all vehicles. RT3D can also render the airplane HUD. A new feature allows the simulation to add text and even simple two-dimensional graphics to the HUD without the need to modify RT3D. The simulation can also control a 3D "path-in-the-sky" feature as a navigational or research aid. As of this writing, RT3D models only cloud-free skies under a noon sun; however, clouds, fog, night conditions and infrared camera views are planned for inclusion in future releases.



080254

Figure 5. RT3D screen capture showing the Ikhana aircraft and HUD.

RT3D is written in C++ using the OpenGL® (Silicon Graphics, Inc., Sunnyvale, California) graphics libraries. Because recent versions no longer require expensive commercial rendering engines, RT3D is now capable of running on most computer platforms and can be freely distributed. RT3D runs under the Linux™ (Linux is a registered trademark of Linus Torvalds), Windows® (Microsoft Corporation, Redmond, Washington), Solaris™ (Sun Microsystems, Santa Clara, California), and Mac OS™ X (Apple, Inc., Cupertino, California) computer operating systems, allowing the same code that is used in a multi-screen panoramic display to also run on the researcher's desktop.

The Earth is modeled as a smooth oblate surface, allowing the curvature of the Earth to be visible at higher altitudes. In selected areas, textured 3D terrain is drawn.

The terrain height field is loaded in 1° by 1° tiles from a USGS digital elevation model with 3 arc-second spacing and a vertical resolution of 1 meter. The data are thinned by RT3D to reduce the number of polygons drawn, thus increasing the frame rate to a reasonable value. Current hardware will run RT3D at approximately 60 Hz with a terrain skip factor of 10 (30 arc-second resolution). Textures for the Edwards or Antelope Valley area were custom-made from cloud-free USGS Landsat 7 images. Textures for the remaining tiles are derived from the publicly available USGS Blue Marble data set. The global texture is derived from the same Blue Marble data set but is down-sampled to approximately 25:1. At higher altitudes, a simple atmospheric fade model allows an accurate star field to be seen. All of these features working together allow RT3D to provide usable visuals up to an altitude of approximately 500,000 ft.

The simulation communicates with RT3D by way of an Ethernet connection. Every graphical object (aircraft, HUD, text packet, et cetera) is represented by its own User Datagram Protocol (UDP) packet stream, and may freely join or leave the shared environment at any time. RT3D can accept input from multiple simulations simultaneously, displaying everything in the shared world with minimal jitter even when the source simulations are running at different rates. One simulation can also drive multiple copies of RT3D, either displaying different views or working together for a panoramic view across multiple screens. RT3D does not send any data back to the simulation.

At times it is useful to have “target” aircraft to chase while evaluating handling qualities. Each simulation has storage for up to nine secondary target aircraft. The simulation can send packets for these aircraft to RT3D and the ground track map for display. Target data can be read from a file, another simulation, or programmed using the Calc language (see section 9 below).

6. REAL–TIME PERFORMANCE

The simulation uses multiple threads of execution to achieve real-time performance. Because the execution order seldom changes, developing a complicated scheduler, as done in other simulation frameworks, was avoided in developing the Core. Instead, the execution sequence of each thread is hard coded as a function. The threads requiring real-time performance are priority-boosted and locked into isolated processors, effectively making the code non-preemptive. Threads for the user interface and various I/O tasks, that do not require real-time performance, are left to the operation system to schedule on any free processors. A mutual exclusion lock (or a similar barrier) is used to synchronize the data exchange between threads.

Timing for real-time threads is achieved by polling on a high-resolution clock until the correct number of nanoseconds has expired. This simple scheme is fairly portable and is more reliable on some systems than using an interval timer. The simulation has facilities to dynamically change the execution rate of each real-time thread, changing the integration step size. The user can also control the ratio of simulated time to actual time, artificially increasing or decreasing the pilot's perceived workload. A C++ class, called SimThread, was created to encapsulate the complexities of creating real-time threads in a machine-independent manner.

The Timer class accesses a high-resolution timer to store timing statistics for each major model in the simulation. This enables the developer to identify where optimization is needed and to properly distribute the workload between real-time threads. These statistics are always available on the TIME display page.

The overall execution rate of the simulation is checked against an external time source. The SimTime class can read time from a variety of IRIG-B time boards into a standard time format. This time is compared against the high-resolution system timer to check for drift. These results are also displayed on the TIME page. The SimTime class can return the current time in a variety of formats when time stamps are needed. A configuration file informs the simulation of what time sources are available on each computer; if a precision time source is not available, the SimTime class uses a combination of system time and the high-resolution timer to provide time stamps. All IRIG-B time boards are synchronized to a single Global Positioning System (GPS) time source, permitting better synchronization between different simulations and reducing jitter in the RT3D graphics.

In addition to the real-time mode, the simulation can run in two other modes. The soft-real-time mode uses the same multi-threaded timing scheme as that used in the real-time mode, but the threads are not locked into dedicated processors and hardware interfaces are not required, permitting most simulations to be "flown" using just a joystick or a mouse computer pointing device on a simple dual-core computer. In batch mode, the simulation is permitted to run as fast as possible. The SimThread class logically combines the real-time functions into a single thread with a simple scheduler to ensure the functions continue to execute at the correct rate relative to one another. There is little I/O in batch mode, thus today's hardware will normally run in batch mode much faster than in real-time mode, making the simulation too fast to be piloted. Batch mode, however, is very useful for scripted tests such as Monte Carlo analysis. Batch mode, as well as the two other modes, can be used with or without the GUI. This differs slightly from the traditional meaning of a batch job. The run mode is selected using a command line argument enabling the same executable to be used for simple batch jobs as well as real-time hardware-in-the-loop simulations.

7. DATA DICTIONARY

Managing data in any large simulation is difficult. Many of the newer simulations, particularly those with accurate modeling of avionics and telemetry, contain tens of thousands of parameters. In an engineering simulation it is expected that each parameter will be well-defined, visible to the user, and available for output to disk and external applications. This section explains how the Core meets these goals.

A data dictionary scheme was developed to manage simulation variables. The definition of each variable in the data dictionary includes, but is not limited to, data type, dimensions, default value, limits, units, sign convention, label line, and detailed description. Dictionaries are plain-text free-format files with syntax similar to that used in the C computer programming language. Variables are grouped into tables, which can be thought of as data structures. In fact, the data dictionary files are read at compile time to produce header files containing C structures and optionally overlaying Fortran common blocks. A tool is available to generate a data dictionary file from a Fortran header file to simplify integration of legacy code.

The compile process also produces code to build an object-oriented symbol table for runtime access to data dictionary information. This symbol table stores all of the information that is known about each variable and provides methods to access the value. The ability to access data by name at runtime is a key feature of the Core. Not only does it give the user full visibility into the simulation, but it is also used to dynamically configure many of the I/O capabilities in the simulation.

The C++ code is not limited to using the C structures defined in the data dictionaries. Each C++ object can also create a uniquely-named symbol table and register the member variables with the table. Data encapsulation is maintained because the C++ object can register methods to get and set the variable's value instead of providing the symbol table with direct access to the variable. The C++ interface to the symbol table does not require the use of the data dictionary, but it does permit attributes (label, units, limits, et cetera) to be loaded from a dictionary file since this syntax is less cumbersome than specifying this information in C++.

The symbol table is implemented using C++ classes with one object created for each published variable in the simulation. This object stores all of the attributes of the variable and provides methods for accessing and displaying this information. The object does not store the variable's value, it stores a reference to data stored elsewhere. A template class is used for numerical variables so that one class can be used to reference all fundamental data types (int, float, double, unsigned short, bool, et cetera). Up to ten-dimensional arrays of these fundamental types are supported. Symbol classes for character arrays and C++ strings are also provided. All symbol classes are derived from the base class Symbol. A container class for Symbol objects (that is, a symbol table) is also derived from Symbol so that the tables can be stored within other tables, creating a hierarchical structure similar to a file system. The GUI has access to all symbol table information using a Java™ class that maps to the C++ Symbol class by way of JNI™ function calls. There is a limited interface to the symbol table for C and Fortran.

8. COMMANDS AND SCRIPTS

The simulation framework has a powerful multi-level command processor. The framework provides global and model-specific command processing, command queuing, nesting of scripts to any level, support for multiple command sources, command recall, and an interpretive language for entering mathematical expressions. The simulation contains a few global commands, but most commands are passed to the currently selected display page for processing. For example, the REC page processes commands related to the data recorder and the CS page processes commands for the control system. Because the CS display is part of the control system model, a different control system with different commands can be substituted without modifying the command processor. Almost all menus, buttons, sliders, and other graphical controls in the GUI are simply shortcuts for sending commands to the appropriate display page.

Since the simulation is command driven, almost all operations can be scripted. A script is predominately a collection of commands saved to a file along with comment lines. Because most modules utilize Calc (see section 9 below) to process command strings, the high-level scripting syntax requires a minimal set of features. There exists an if-then-else syntax to select between different command blocks. The script processor also provides argument substitution similar to a C shell script, allowing the same script to be used for multiple tests. For example, a script can be written to input a pitch stick doublet and to record the results; arguments following the script name can set the doublet's amplitude and width.

The simulation does not have a built-in plotting capability; however, a system call can be added to a simulation script to run an external plotting application. Since it is convenient to use the same argument substitution in the plot script as the simulation script, the simulation command processor was modified to permit scripts for other applications to be embedded into simulation scripts. Therefore, the initial conditions, test setup, data recorder setup, test execution, and the plotting commands can all be contained within a single script. The user can select a script to run and see the results plotted within a few seconds. This method works almost as well as a built-in plotting tool.

9. THE CALC LANGUAGE

As mentioned above, each module of the simulation is responsible for processing commands related to that module. Much of this input is mathematical expressions used to set variables within the module. An interpretive programming language, called Calc, was developed to help the module developer parse and evaluate this input. Although other interpretive programming languages are available, by writing an application-specific language we can provide full access to the simulation's symbol table, utilize preferred syntax, add new features as needed, and avoid portability and licensing issues. Calc is written in C++.

A module developer can use Calc to either evaluate an input string immediately returning a double or a string result (depending on the input statement) or use it simply to parse the input returning a tree structure that can be evaluated at a later time. The latter is used for efficiency when input needs to be reevaluated repeatedly.

It is critical to prevent the user from entering commands that can crash the simulation, so Calc performs extensive error checking while parsing and evaluating expressions. Unlike most compiled languages, Calc checks for runtime errors such as index-out-of-range, divide-by-zero, and invalid function arguments (for example, acos(2)).

Calc has full access to all variables in the simulation symbol table and to their attributes such as minimum, maximum, and default values. Calc is a curly-brace programming language with syntax similar to that of C, C++, or Java™ and includes all of the mathematical, assignment, logical, bitwise, increment, and comparison operators in the C language. Calc supports C-style block if-statements, for-loops, and while-loops with break and continue statements. Although Calc does not yet permit user-defined functions, it does include many built-in functions for both numerical and string operations. All numerical operations except bitwise operations are performed using double-precision mathematics. If a variable in the simulation is not a double, the symbol table will automatically cast it to a double before Calc uses it. The symbol table also manages the conversion back to the native data type when setting a variable. Strings in Calc are handled in a manner similar to the method utilized for the C++ std::string class, including many of the same string operations. The Calc parser handles both C and C++ style comments. The reason for an interpreter of this complexity will become apparent below, during the discussion of the auto-test module.

10. INITIAL CONDITIONS

There are three simulation run modes: reset, operate, and hold. In reset mode, models are executed but the equations of motion are not integrated. In operate mode, time increments and the equations of motion are integrated. In hold mode, none of the models run, effectively pausing the simulation.

In reset mode, the user can initialize the airplane to any flight condition. To avoid a large transient when the mode is switched to operate, the forces and moments on the airplane, at the new flight condition, can be trimmed out. There are several options for trimming the airplane. The user can select from constant velocity, constant alpha, constant thrust, or stick-only trim while maintaining straight and level flight. The aircraft can also be trimmed to a coordinated constant *g* right or left turn.

Initial conditions are stored in an aircraft state vector. More often than not, the initial conditions will not be the variables with which the user will want to work. Thus, when the user enters a new initial condition, such as altitude or Mach number, a new state vector must be calculated. This illustrates why functions, not just variables, need to be registered within the symbol table. If, for example, the user enters a new Mach number initial condition (IC.mach) the symbol table will invoke a function to set the complete aircraft state vector instead of setting only IC.mach. The aircraft state vector can be saved at any flight condition, enabling the simulation to be reinitialized to that point.

The simulation can load the location and orientation of runways from a data file, normally the same runways that will be loaded into the visual system. When the airplane approaches one of these runways, the terrain altitude is set to the runway altitude to smooth the ground surrounding the runway. The runway module provides an easy way to line up to a runway for takeoffs and landings. The user can specify the altitude, downrange, and crossrange distance of the airplane relative to a runway for both takeoffs and landings. When a runway is selected for a takeoff or landing, the airplane's initial conditions are computed to match these offsets.

Another very useful new feature is the ability to set the airplane's initial conditions to track another aircraft. As with runways, offsets can be set between the simulated airplane and the "target" airplane. When enabled, the simulation's initial conditions will be updated to follow the target until the simulation is switched into operate mode. This saves much time during setting the initial conditions of multiple simulations for formation flight. The new Map/Radar display in the GUI, seen in the top of figure 4, shows the location of target aircraft and runways relative to the simulated airplane.

11. SIMULATION LOG

The simulation maintains a log of all commands entered during a session and any responses from the simulation. Each log message is time-stamped with both the wall-clock time and the simulation time. The source of each message (script, user input, sim, et cetera) and the type of each message (comment, command, warning, error, Calc input, et cetera) are also attached to each message. The log is saved to disk after each message, so in the highly unlikely event that the simulation crashes while processing a script, the user can determine the last command executed.

The log is displayed as part of the main window, shown in the lower left portion of figure 3, or optionally in a larger standalone window. The GUI has options for saving, clearing, and sorting the log window, and permits commands to be executed again by clicking on them with the mouse pointer. Because the user may have changed the display page since the command was last entered, the type tag saved with the command is used to ensure the command is routed to the correct module. A group of commands can be highlighted and saved to a file to create a script.

There are three types of message output from the simulation to the log: informational, warning, and error. Warnings are displayed in magenta; errors are displayed in red. The Core provides functions to write these three types of messages to the log from C++, C, Fortran, Java™, and Calc. The Calc, C++, and C functions use a printf() style format string with a variable-length argument list for easy formatting of these messages.

12. FILE INPUT/OUTPUT

NASA Dryden has developed a variety of formats for storing compressed and uncompressed time-history data. Collectively these formats are referred to as the Dryden Time History (DTH) formats. These formats include the GetFDAS data formats, which are used at NASA Dryden to store flight-test data time histories. Some of the formats are directly compatible with Microsoft Excel® (Microsoft Corporation, Redmond, Washington). A software library is available to read and write these formats. Data analysis tools developed in-house, as well as commercial products such as MATLAB® (The MathWorks, Natick, Massachusetts), can read DTH data using these library functions. The Core also uses the DTH libraries to read and write time-history data.

The user can specify the simulation parameters that are to be recorded by listing the variable names in a setup file or by dynamically adding them on the REC display page. The recorder permits new variables to be created by entering Calc language expressions. This is useful for changing units or simply to rename a variable before recording. In addition to the file format and the parameter list, the user is provided with a number of other options to control how and when the recorder is started and stopped and at what rate the data are recorded.

The recorder buffers output in the main real-time thread of the simulation to ensure the data are frame-coherent. Writing to disk, however, is normally too slow be included in a real-time thread, so data are written to a circular memory buffer and then written to disk using a separate thread of execution. Unless the system is highly loaded, this scheme will permit large sets of frame-coherent data to be written to disk at the full simulation frame rate without dropping any data.

The simulation can also read time-history data in real time. This capability enables reading pilot inputs recorded during a flight test so that simulation data can be compared to flight data, reading in a prerecorded trajectory for a target aircraft, or using flight data to drive the inputs to a particular model such as a flight-loads model.

The data input module creates a temporary variable for each parameter found in the input file, updating these variables whenever the simulation time becomes greater than or equal to the next time stamp in the file. The data input (INP) display page provides several options for controlling how the simulation time is compared to the file time. There is also an option to use linear interpolation on the input data. To utilize the input data, the user provides a block of Calc-language-interpreted code that sets simulation variables from the temporary variables. This code can be evaluated at any test-point (see section 13 below) permitting the file data to override the simulation data at the correct location in the simulation's execution sequence. The extra step of using temporary variables is to permit name and unit conversions from the recorded data. It also prevents extra data in the input file from unintentionally overriding simulation data. The data input module uses a multi-threaded, buffering approach similar to the data recorder to permit large sets of data to be read in at the full simulation frame rate without dropping any records or causing frame overruns.

13. AUTOMATED TESTING

Perhaps one of the more unusual features of the Core is its capability for automated testing. The scripting, time-history I/O, and the simulation log features already discussed provide considerable capabilities for automating testing. These tools, however, assume that the simulation developer had the foresight and the time to program into the simulation every conceivable test condition, failure mode, and diagnostic. Instead of burdening the developer with this monumental task, we have developed a generic method that a simulation engineer or end-user can utilize to enter test conditions, faults, and diagnostic messages at runtime. This section discusses three additional features for automated testing: quick-check, auto-test, and messaging.

The quick-check capability is used in scripts to perform a quick one-time test on a variable's value and to write a passed or failed message to the program log. For example, the input “pla<=pla.max” will result in the message:

```
PASSED: (pla<=pla.max)  values: pla=75  pla.max=100
```

if pla is less than or equal to pla.max, or the message:

```
FAILED: (pla<=pla.max)  values: pla=102  pla.max=100
```

if pla is greater than pla.max. Any Calc language logical operators (==, !=, <=, >=, &&, ||, !), math operators, and functions can be used in the test condition. To avoid the common error of using “=” instead of “==” in the test expression, assignment statements are not permitted. The quick-check capability is very useful in regression testing, since a single input may propagate into multiple models, over multiple avionics buses, and into multiple telemetry streams. A script can be written to set an input and check to make sure the change occurs correctly in all of its forms.

The auto-test feature permits the Calc language code to run at predetermined locations in the simulation's execution sequence. These locations are called test-points and are normally placed before and after every major model in the simulation. Since Calc is an interpreted language, code can be added and removed at runtime. Most error conditions can be simulated with a single line of Calc code. For example, Calc can be used to freeze, bias, or add noise to an actuator simply by overriding the actuator position at the test-point just after the actuator model. Furthermore, if-then-else logic can be used such that the error condition only triggers at a given time or when a specific event occurs. The input to add noise to the left aileron for 20 seconds starting 2 minutes into the flight might look like this:

```
if (t>120 && t<=140) ail_left += 0.5*noise(1);
```

The noise() function returns a random number between -1 and +1; the argument is the seed value. Functions also exist to easily add ramps, square waves, sine waves, and frequency sweeps. This eliminates the need to model many possible malfunctions, and many users find the syntax easier to deal with than learning and setting up all the options for a hard-coded malfunction.

There are times when hard-coded logic is preferable, for example, failing an actuator to a trail-damped mode in which the aerodynamic forces affect the actuator position. Even in this case, the malfunction implementation is simplified because the modeler only needs to implement the failure model; the triggering logic can still be performed using the auto-test feature.

The auto-test feature can be used to program pilot and other inputs in addition to malfunctions. Typically, each simulation has a set of scripts that input stick, rudder, and throttle commands at various flight conditions using the auto-test feature and over-plot the results with data from the baseline simulation for comparison. Users have discovered many other uses for the auto-test feature, such as programming a wing leveler, altitude hold, and target aircraft trajectories. In many cases, a few lines of auto-test code prevent the lengthy process of hard-coding a new feature into the simulation.

The messaging feature can be thought of as a combination of the quick-check and auto-test features. Messaging permits diagnostic tests to be repeatedly evaluated at selected test-points. Although the Calc language can be used at test-points to write info, warning, and error messages to the simulation log, a repetitive test will quickly flood the log window. Thus a second form of the info, warning, and error messages as added to write to a message stack instead of the log. The top left of figure 3 shows an example of the message stack. The stack only displays each message once regardless of how often the test condition is evaluated to true. For example, the auto-test code:

```
if (atrm) info(“Auto-trim is on”);
```

will continually write to the simulation log while atrm is true. The code:

```
Info(atrm, “Auto-trim is on”);
```

will cause the message to appear once on the message stack (the MSG display page) while atrm is true. The message will be cleared a short time after atrm becomes false. The delay in clearing is to prevent messages from flashing across the screen too quickly to be read by the user. Each message is also written to the log the first time the test evaluates to true, and a message summary is logged just before the simulation exits.

There are three forms of the message function: Info(), Warning(), and Error(). These control the color and sorting options in the message stack. The first argument to each of these functions is the test condition, which is very flexible as with the quick-check test condition. The second argument is a format string for the message. Optional arguments follow the format string using C printf() style formatting. Since all Calc language code is pre-parsed thousands of diagnostic messages can be added at runtime without impacting the real-time performance of the simulation.

14. MISSION CONTROL CENTER INTERFACE

Some of the simulators at NASA Dryden can be interconnected with the Mission Control Center (MCC) as described in reference 8. This permits pilots and engineers to rehearse flight tests, train control room staff, and test control room displays in advance of flight missions. The interconnections between the simulators and the MCC include audio, video, radar, and telemetry (see fig. 2). Radio traffic is handled by providing the pilot in the simulator with a link to the control room communications system. Video from the HUD camera and long-range optics are simulated by running the RT3D application and converting the video signal from the image generator into a television signal. Radar tracking data are output from the simulation computer on an RS-232 line. Telemetry data are generated from the simulation using a pulse code modulation (PCM) encoder. Fiber optics is used for the transmission of the data between buildings.

The only connection that represents a significant challenge to generate is the telemetry stream(s). The first step is to map simulation variables to the variable names and units used in the telemetry. Although much of the sensor data in the telemetry are already modeled for the avionics buses, the simulation will still typically require expansion to provide data for strain gages and other sensors normally not required by the simulator itself. The software provides a way to insert constant nominal values when detailed modeling of these sensors is not required.

The next step is to convert the engineering data into raw telemetry counts (scaled integers). Polynomial coefficients are normally provided to convert from counts into engineering units. The simulation needs to convert from engineering units to counts. If the polynomial cannot be inverted, then the simulation uses a Newton-Rapson iteration to solve the polynomial and obtain raw counts (ref. 9). These raw data are then written to a buffer in high-speed reflective memory.

A program running on a VME-based single-board computer reads the raw telemetry data from the reflective memory and populates the telemetry map on a hardware encoder. This software is developed in-house and runs under the VxWorks real-time operating system. The configuration files for both the engineering-unit-to-counts converter and the frame encoder are generated by a setup program from a telemetry attributes transfer standard (TMATS) definition file. Future simulators will have the telemetry encoder installed directly into the simulation computer, eliminating some costly intermediate hardware.

The simulation laboratory is also equipped with a telemetry front-end processor compatible with the system used in the MCC. Not only is this system necessary to pre-check the telemetry stream for proper encoding, it is also used to drive control room displays located in the simulation laboratory.

15. REAL–TIME ETHERNET SERVER

Because of the hardware cost and the time involved to set up a telemetry encoder and decoder system, the simulation needed a more direct path to drive control room applications. This was accomplished by having the simulation generate the Ethernet data packets that would normally be sent from the telemetry front-end processor. This module of the simulation is called the real-time Ethernet server (RES).

Most of the Mission Control room applications read data from a current value table (CVT) in shared memory. This memory is populated when the host computer receives a RES data packet. A CVT file specifies the offset and data type for each variable in the table. The simulation uses a modified version of this file that contains an extra column to define a mathematical expression that the simulation uses to compute the CVT parameter. This expression is entered in the Calc-interpreted language so there is no need to recompile the simulation to change the output data stream. The data packets can be sent from the simulation to multiple control room applications using a UDP broadcast address. RES supports byte swapping in the output packet if needed, and the output rate can be set by the user.

The disadvantage of outputting an RES stream directly from the simulation instead of using the telemetry front-end processor is that the code for generating derived parameters, which normally runs on the front-end processor, may need to be replicated inside of the simulation.

16. EXTERNAL APPLICATION INTERFACES

In addition to the RES interface, the Core provides generic bidirectional interfaces to external applications. The Talk feature permits external applications to exchange data with the simulation through a local shared memory region and the TCP_IO feature uses an Ethernet connection for data transfer. Each method has its own advantages and disadvantages, but both methods permit periodic data exchange between the applications, queries of the simulation's symbol table, queuing of simulation commands, and the choice of synchronous or asynchronous execution.

The Talk feature utilizes the auto-test feature in the simulation to copy data between simulation variables and data arrays in a shared memory region. Any external application that can call a C function can read from and write to these shared memory arrays. MATLAB® uses a MATLAB-executable (MEX) function to exchange vectors of data with the simulation in this way. The Talk feature has been used to run Simulink® (The MathWorks, Natick, Massachusetts) control system models as if the model were a function called from the simulation’s real-time loop. Other uses include scripting the simulation using MATLAB® M-files and using Simulink® to produce real-time strip-charts of simulation data. Although simple to use, the Talk feature has a few limitations; most notably that the external application must run on the same computer as the simulation and that only one external application can use the interface at a time.

The Core also provides an Ethernet interface without these limitations. This feature is called TCP_IO because it utilizes the TCP/IP network protocol. The TCP_IO feature enables the simulation to operate as an Ethernet server. The client application can attach to the server and configure the connection to send and receive periodic data packets. The TCP_IO feature also gives the client application the ability to send command strings to the simulation, and provides an extensive set of symbol table queries. The server does not limit the number of client applications that can connect to the simulation, and the server provides password protection and other security precautions.

The connection between the simulation server and client application can be synchronous or asynchronous. A synchronous connection will cause the client to run in lockstep with the simulation, exchanging data at the specified test-point. Because a synchronous connection can inhibit real-time operation, an asynchronous mode was added. In the asynchronous mode the data packets are processed at the specified test-point, but the packets are transmitted and received using a separate thread to avoid possible frame overruns. Double buffering is used to ensure packets remain frame-coherent.

Each simulation also has a client module that can connect to multiple server simulations at multiple test-points. The client module permits the two simulations to exchange any data periodically or on demand and is completely reconfigurable at runtime. This flexibility permits the simulations to interact for a variety of reasons such as autonomous formation flight experiments or captive carry.

A small set of C++ (or Java™) classes is provided for building additional client applications. One notable feature of this package is that it permits local variables in the client application to be mapped to simulation variables. Once the mapping is complete, the client only need call one function to synchronize data between the two applications. The data exchange, data type conversions, and unit conversions are performed automatically. Several clients have already been developed using these classes. A MEX function was written for MATLAB® and Simulink® and Java™ touchscreen control panels have been developed.

There are two other more indirect ways in which MATLAB® and Simulink® have been used with the simulation. The Talk and TCP_IO interfaces work well during model development. Once a control system model is functional, Simulink® can generate C code that is compiled into the simulation. This has obvious advantages over linking the two programs, such as ease of use, better source code control, and real-time performance. Another tool used in control system development is to generate a linear model of the aircraft for use in Simulink®. The simulation has the ability to perturb aircraft states and control surface deflections to generate matrices necessary for creating a linear model. These matrices can be output in the MATLAB® M-file format.

17. DATA BUS TOOLS

MIL-STD-1553 is a specification for a digital, transformer-coupled, dual-redundant bus that permits flight computers to communicate using a simple message-passing protocol. Each flight component connected to the bus is assigned one or more remote-terminal number(s) that uniquely identify it on that bus. Each remote terminal can contain up to 32 transmit and 32 receive message buffers (sometimes called the subaddress). Each subaddress can contain up to 32 16-bit data words. Each bus has a bus controller that determines when data are transmitted across that bus. Typically, the mission computer or one of the flight control computers acts as the bus controller. The transfer rate on a MIL-STD-1553 bus is approximately 1Mbit/s.

Obviously, this bus design would be very limiting if each discrete was transmitted as a 16-bit word, or every floating-point value as four 16-bit words (one 64-bit word). For this reason data are normally "packed" into the raw message buffer. One 16-bit word can hold 16 discrete values, two 8-bit integers, a 16-bit integer, a scaled floating-point number, or some other compressed data format. This scaling and packing of data into a message buffer is handled by a pack function. After the message is transmitted to its destination it has to travel through the reverse process. The reverse process is handled by an extract function. Because of latency during the transfer process, the simulation must keep two data structures for each bus message: one data structure for the transmitter and one for the receiver. These message structures contain status information that is filled in by the hardware interface, the raw data, and the extracted engineering unit data for the message. Figure 6 depicts the process of packing, sending, and extracting data for a remote-terminal to bus-controller transfer.

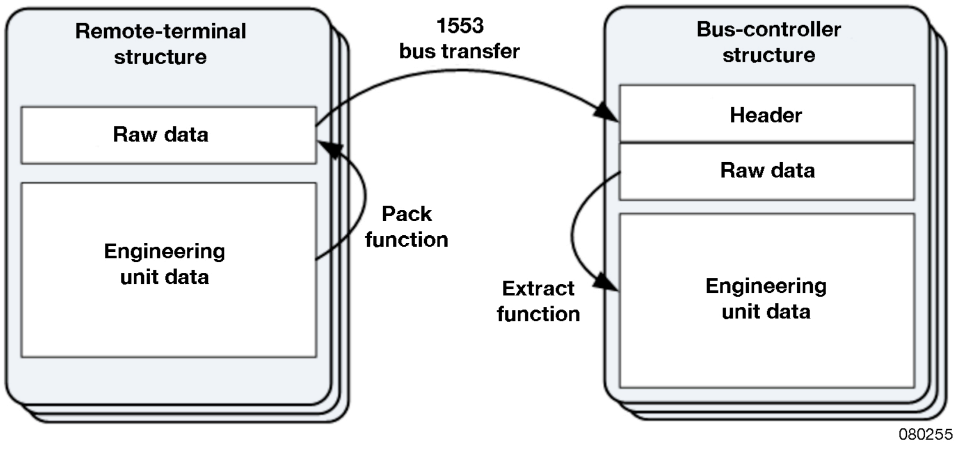


Figure 6. Remote-terminal to bus-control data transfer.

The process of coding the message structures and the routines to pack and extract the raw data is time-consuming and prone to error. Fortunately, most aircraft have a digital database containing all the information necessary to automatically generate the required code. A simple database query or reformatting a spreadsheet can convert the bus database into a properly-formatted plain text file. The simulation reads this text file at compile time and generates the message structures as well as the pack and extract functions. The code generator supports almost any conceivable way of bit-packing messages including multiplexed data. Since the code generator produces error-free code, the model developers can focus on writing the logic in the avionics instead of the hardware interfaces.

MIL-STD-1553 hardware can now simulate multiple remote terminals and the bus controller, and monitor all bus traffic for multiple buses on a single computer board. The Core provides C++ classes to interface with this hardware using a different derived class for each different hardware interface. A configuration file informs the simulation of what hardware is available in each computer so the correct bus interface objects can be instantiated. Remote-terminal output data are normally updated each simulation frame by copying the raw message buffer to the correct location on the hardware. The bus-controller chain (the command words and outgoing data) is built in local memory and block-transferred to the hardware for efficiency. Because some aircraft require synchronization of messages across multiple buses, starting the bus chain is separated from the transfer of the bus chain. The simulation does not poll for incoming remote-terminal and bus-controller data. Instead, the hardware is configured to sequentially monitor all bus traffic. The sequential monitor buffer is block-transferred to local memory and parsed for each simulation frame. This is efficient because only the messages found in the buffer need to be converted to engineering unit data.

The base class for the hardware interfaces, called Bus1553, can simulate the remote-terminal, bus-controller, and sequential-monitor features of the hardware when a hardware bus is not available, such as when running the simulation in batch mode. This ensures that the data used in the avionics models have the same quantization and latency effects as if a real bus were used.

Because modeling avionics is a major part of today’s simulations, the Core has several built-in tools with which to view and analyze the bus traffic. One feature converts the data from the sequential monitor into a plain text format that can be viewed in a text editor. A flexible set of filters can be used with this feature, such as only recording messages that contain errors. The GUI also provides a real-time message viewer that shows the raw or engineering unit data for both the transmit and receive message structures. The BUS display page provides summary statistics for all buses and has options to control what subsystems are simulated or monitored. Of course, the auto-test and time-history I/O functions discussed above are often used to analyze bus traffic. Finally, the simulation outputs all bus traffic to a circular buffer in shared or reflective memory for access by external applications.

The Core also supports the ARINC-429 bus interface. Because of the minimal amount of data on these buses, the packing and extracting of data is handled using Calc language expressions read from a configuration file instead of using a C++ code generator as is done with the MIL-STD-1553 bus.

18. CONCLUDING REMARKS

The National Aeronautics and Space Administration Dryden Flight Research Center has developed a new “Core” software framework for fixed-wing aircraft simulation. This framework provides researchers with new and enhanced tools for training and analysis tasks, helping to mitigate the risks associated with flight research. The new, more modular architecture and code generation tools also help minimize the development time needed to create new simulations. Additionally, the Core provides a consistent user interface and improved documentation, reducing the learning curve and making it easier for researchers to move from one flight research project to another.

The success of the Core is largely due to its evolutionary, as opposed to revolutionary, approach. By maintaining key features such as support for multiple programming languages, runtime access of data, and the display page concept, the effort involved in integrating Core features into legacy simulations has been minimized. In the future, the Core will continue to evolve to meet the needs of the flight research environment.

APPENDIX A

DISPLAY PAGE LIST

This appendix provides a typical list of display pages found in a NASA Dryden Flight Research Center simulation.

General Display Pages	
ALL	Summary of the most important aircraft states.
GUST	Used to configure discrete wind gusts in the aircraft coordinate system as a function of time.
IC	Sets the initial conditions such as airspeed and altitude.
MAP	Used to enter and view flight plan data and sets options for the Map/Radar graphic. Also used to set up the RT3D path-in-the-sky feature.
MENU	List of all display pages and global commands.
NOTE	Used in scripts to display a message to the user.
PC	Program Control—sets operational limits on the simulation. When a limit is reached the simulation goes into hold mode and a warning is logged. Can be used to step through the simulation frame-by-frame.
RWY	Used to set the aircraft initial conditions for takeoff or landing on a selected runway. Crossrange, downrange, and altitude relative to the end of the runway can be specified.
TIME	Displays simulation frame and model timing statistics and sets options that control the simulation execute rate.
TURB	Random turbulence model with selectable severity level.
TRIM	Controls how the simulation is trimmed while in reset mode.
VAR	Displays a list of user selected variables with units and labels. Also, permits searching the symbol table using regular expressions. The GUI provides graphical window with similar features.
WIND	Used to set the wind profile based on altitude.

Data Input/Output Displays	
BUS	Displays MIL-STD-1553 bus data given the bus, remote-terminal, and sub-address numbers. Provides a good summary of bus statistics but the graphical message viewer is usually a better tool for viewing data from the bus.
CKPT	Used for debugging cockpit or joystick I/O problems. Some simulations split this into two pages; one for the cockpit (CIU) and one for the stick (SES).
INP	Sets up and controls reading of time-history data into the simulation. For example, pilot inputs recorded in flight can be loaded into the simulation to compare the simulation's response to flight data.
LIN	Used to generate a linear model that can be loaded in MATLAB® or Simulink® for control system development.
PCM	A few of the simulators are equipped with hardware to generate a telemetry stream that can be sent to the control room for training purposes. This display page controls this interface.
REC	Real-time data recorder controls. Can output data to any of the Dryden time-history file formats. Signals to record can be specified on the display page or loaded from file. Permits new signals to be created for recording purposes.
RES	Real-time Ethernet server - used to broadcast select data to control room display applications.
RT3D	Interface to the RT3D out-the-window graphics application. It is normally easier to use the graphical controls in the GUI to operate RT3D but this page contains some features not on the graphical panel and is required when scripting graphics setup.
TALK	Provides a shared memory interface between the simulation and external applications such as MATLAB® and Simulink®.
TCPC	The TCP client page is used to set up a network connection for data exchange with another simulation running the TCP server.
TCPS	The TCP server page is used to start the simulation network server permitting client applications such as MATLAB®, Simulink®, remote displays, and other simulations to exchange data with the server simulation.
TEXT	Used to configure text and simple two-dimensional shapes to be sent to RT3D for display. The TEST page can be used to turn on and off these custom messages.
TRGT	The target page provides storage for position and attitude data of secondary aircraft that can be output to RT3D and ground track displays. Target data can be loaded from a file, generated from a TEST page, or read in from another simulation.

Test Pages	
TEST	Provides a mechanism for the user to enter test conditions (faults, diagnostic messages, et cetera) into the execution stream of the simulation.
MSG	The message page works in conjunction with the TEST page to display diagnostic messages on a color-coded message stack.
FREQ	Utilized to set up and control the frequency sweep generators.
QCHK	The quick-check page is used in scripts to perform a quick one-time check on a variable’s value (or a more complicated expression) and write a PASSED or FAILED message to the log window.

Aircraft-Specific Pages	
ACT	Actuator data.
AERO	Aerodynamic data.
CD	Center of gravity, weights, and inertia data.
CS	Control system inputs, outputs, gains, filter coefficients, and limits.
ENG	Engine performance data.
GEAR	Landing gear model data.

REFERENCES

1. Waltman, Gene L., *Black Magic and Gremlins: Analog Flight Simulations at NASA's Flight Research Center*, NASA SP-2000-4520, 2000.

2. Smith, John P., Schilling, Lawrence J., and Wagner, Charles A., *Simulation at Dryden Flight Research Facility From 1957 to 1982*, NASA TM-101695, 1989.

3. Mackall, D. A., Pickett, M. D., Schilling, L. J., and Wagner, C. A., *The NASA Integrated Test Facility and Its Impact on Flight Research*, NASA TM-100418, 1988.

4. Norlin, Ken A., *Flight Simulation Software at NASA Dryden Flight Research Center*, NASA TM-104315, 1995.

5. Clarke, Robert, Lintereur, Louis, and Bahm, Catherine, "Documenting the NASA DFRC Oblate Earth Simulation Equations of Motion" (NASA internal memorandum).

6. Clarke, Robert, "Documenting the NASA DFRC Simulation EOM Integration Algorithm" (NASA internal memorandum).

7. Cotting, M. Christopher, and Cox, Timothy H., *A Generic Guidance and Control Structure for Six-Degree-of-Freedom Conceptual Aircraft Design*, NASA/TM-2005-212866, 2005.

8. Shy, Karla S., Hageman, Jacob J., and Le, Jeanette H., *The Role of Aircraft Simulation in Improving Flight Safety Through Control Training*, NASA/TM-2002-210731, 2002.

9. Fantini, Jay A., *Conversion From Engineering Units to Telemetry Counts on Dryden Flight Simulations*, NASA/CR-1998-206563, 1998.

REPORT DOCUMENTATION PAGE			*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE *(DD-MM-YYYY)* 01-10-2008	**2. REPORT TYPE** Technical Memorandum	**3. DATES COVERED** *(From - To)*
4. TITLE AND SUBTITLE A Software Framework for Aircraft Simulation		**5a. CONTRACT NUMBER**
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Brian P. Curlett		**5d. PROJECT NUMBER**
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) NASA Dryden Flight Research Center P.O. Box 273 Edwards, California 93523-0273		**8. PERFORMING ORGANIZATION REPORT NUMBER** H-2880
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) National Aeronautics and Space Administration Washington, DC 20546-0001		**10. SPONSORING/MONITOR'S ACRONYM(S)** NASA
		11. SPONSORING/MONITORING REPORT NUMBER NASA/TM-2008-214639

12. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified--Unlimited
Subject Category 05 Availability: NASA CASI (301) 621-0390 Distribution: Standard

13. SUPPLEMENTARY NOTES
Curlett, NASA Dryden Flight Research Center. An electronic version can be found at http://dtrs.dfrc.nasa.gov or htttp://ntrs.nasa.gov/search.jsp.

14. ABSTRACT
The National Aeronautics and Space Administration Dryden Flight Research Center has a long history in developing simulations of experimental fixed-wing aircraft from gliders to suborbital vehicles on platforms ranging from desktop simulators to pilot-in-the-loop / aircraft-in-the-loop simulators. Regardless of the aircraft or simulator hardware, much of the software framework is common to all NASA Dryden simulators. Some of this software has withstood the test of time, but in recent years the push toward high-fidelity user-friendly simulations has resulted in some significant changes. This report presents an overview of the current NASA Dryden simulation software framework and capabilities with an emphasis on the new features that have permitted NASA to develop more capable simulations while maintaining the same staffing levels.

15. SUBJECT TERMS
Flight Simulation, Simulation Software

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19b. NAME OF RESPONSIBLE PERSON** STI Help Desk (email: help@sti.nasa.gov)
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			**19b. TELEPHONE NUMBER** ***(Include area code)***
U	U	U	UU	36	(301) 621-0390