Steam Locomotive Cab Simulator
aka: My Ultimate Toy Train (MUTT)
Part 2 of 2 - Operation

Background

Part 1 (Link 1) summarized the design and construction of MUTT, which began in September, 2003. What follows is a summary of finally achieving full operation of the simulator, including tailoring of the computer hardware/software configuration and considerations for future design of simulators. A lot has happened in the train simulation software world since Part 1 of this article was posted in January, 2007. Kuju released *Rail Simulator*, now being maintained by RSDL. Microsoft began development of a second version of *MSTS* based on their *Flight Simulator X* and *Vista* architecture, but subsequently suspended it in January, 2009 as a victim of U.S. economic downturn. Auran released additional versions of *Trainz*. PIE/RailDriver released some new X-Keys hardware, but no news regarding their suspended *TMTS* or release of their long-awaited ReDAC Gauge Module.

As stated in Part 1, one major limitation of Microsoft's original *MSTS* is the lack of an Application Program Interface (API). Had *MSTS-2* been completed, it would have afforded *SimConnect,* which is what the flight simulation community uses for interface to cockpit hardware. Unfortunately, *MSTS-2* was not to be and to date, none of the other current train simulation released products have provided this capability.

Then, thanks to the suggestion of another cab simulator enthusiast, I discovered *Cheat Engine* (*CE*) *5.5* (Link 2). *CE* is a freeware tool used to locate and modify parameters within a game program to gain various advantages during play. I quickly found it to be a well-programmed, powerful tool that provided the key for unlocking the mystery of establishing a dynamic interface with *MSTS*. The sections that follow in this article summarize how the interface was finally established, the operational result in the cab, and my thoughts on the prospects for what is now proven to be possible.

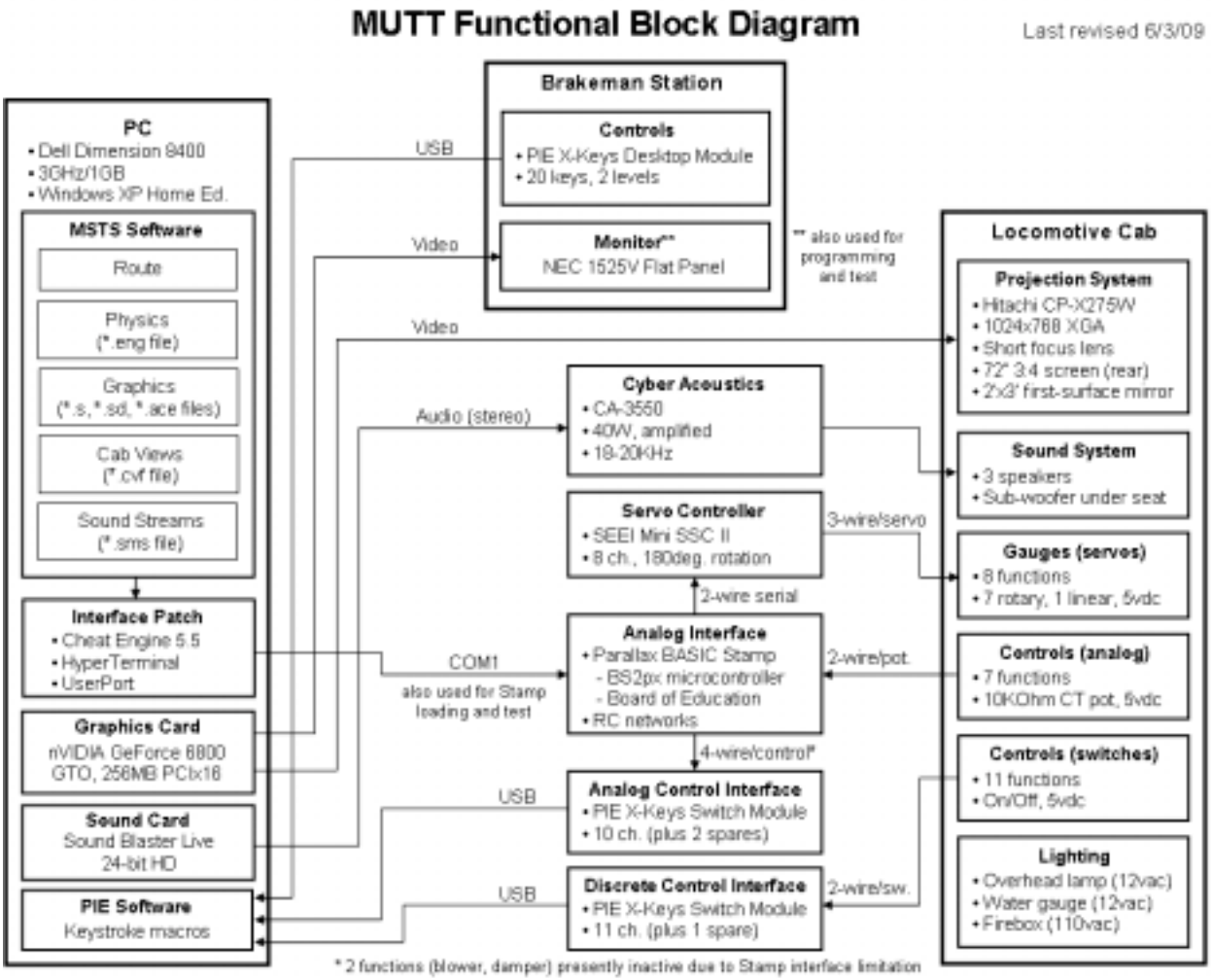
Interface Development

Use of *CE* consists of three basic procedural steps:

1. **Scanning** the game program to discover the location of a selected variable parameter and then discovery of the program instruction(s) that modifies or accesses that location. This was the most significant and time consuming of the three steps.
2. **Programming** a patch to replace the game instruction with a branch to new instructions that capture the variable parameter of interest and perform operations with it. The patch code is in x86 assembly language with an option to insert scripts developed in higher level languages. I opted to use assembly language to keep things simple.
3. **Applying** the patch to the game at run time – the easiest part of the job.

One extremely beneficial programming feature of *CE* is that multiple individual patches can be imbedded within the main patch file, including the ability to share data parameters between the individual patches. For example, the *MSTS* patch file for MUTT consists of 16 individual patches – essentially one for each interface variable. I'll leave the description of *CE* at that. The web site (Link 2) offers an excellent tutorial that should be followed by first time users.

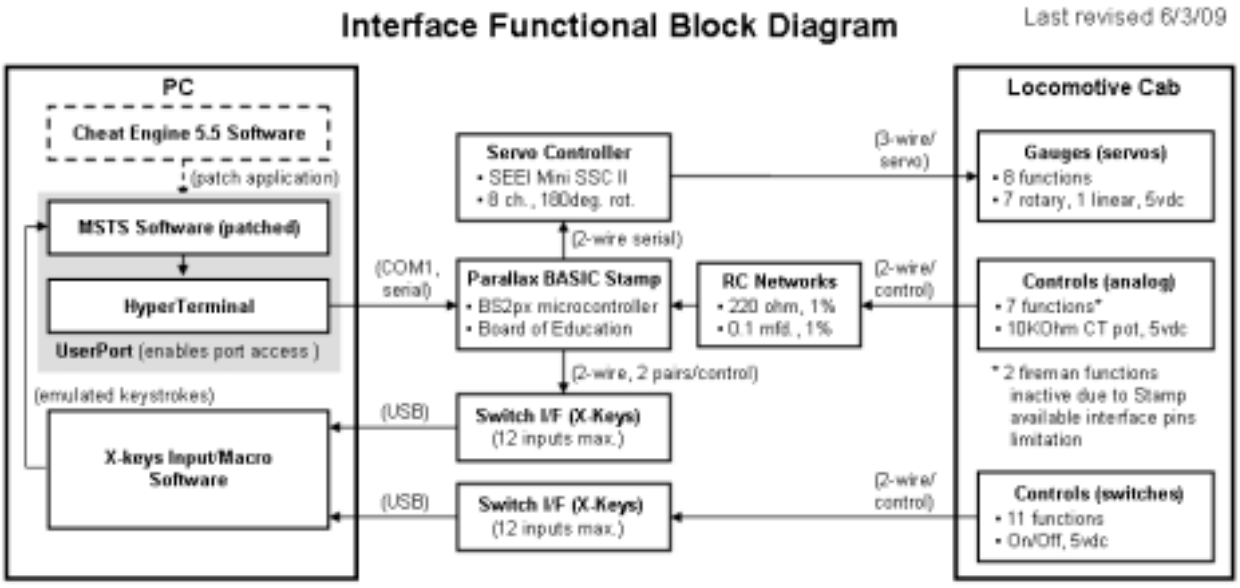
The simulator architecture is represented by the following MUTT Functional Block Diagram.



The locomotive cab itself has changed very little since Part 1 of this article. It was completely disassembled for a residence move and reassembled as shown in the following photos.

Details of MUTT hardware construction were presented in Part 1 of this article. The only major change since Part 1 was the addition of a brakeman's station outside the cab (shown in the photo at the right). It includes a key pad to perform handbrake, uncoupling, and turnout operations.

Part 2 will focus on the hardware interface aspects for gauges and controls, which are shown in the following Interface Functional Block Diagram.



The following is an example of "servo loop" logic used within the Parallax Stamp external processor to track a cab analog control:

```
' Initialize Regulator keyboard command states (keys A and D)
  REGULATOR _CLOSE = FALSE
  REGULATOR _OPEN = FALSE
' REGULATOR is the current simulator control setting
' REGULATOR_CAB is the current cab control position (after scaling)
' Determine if current simulator state is more closed than cab control position
  IF REGULATOR - REGULATOR_CAB) < -DB THEN REGULATOR _OPEN = TRUE
' Produces "D" keyboard command to MSTS
  GOTO END
' Determine if current simulator state is more open than cab control position
  IF (REGULATOR - REGULATOR_CAB) > DB THEN REGULATOR _CLOSE = TRUE
' Produces "A" keyboard command to MSTS
  END
' Output Regulator discrete command states to X-Keys Switch Interface
```

Note that the example employs a deadband (DB) to reduce "hunting"

Discrete controls (on/off switches) are the simplest interface and utilize a PIE Switch Interface module (Link 3) which provides programmable keystroke macros, including combinations and sequences of keystrokes. All other interface is through an external microcontroller, a Parallax Stamp BS2px (Link 4), which receives *MSTS* parameters via the COM1 serial port. Asynchronous communication is used to avoid having *MSTS* wait for a request for data, thereby preserving frame rate. For gauges, the Stamp receives each parameter value, performs calibration offset and scaling, and sends the result to a SEEI Mini SSC II servo driver card (Link 5). For analog controls (proportional) the Stamp "reads" the timed output of a precision RC network to determine the physical control position in the cab, compares it to the received *MSTS* setting, and issues increase/decrease/hold states to another PIE Switch Interface module which in turn issues the appropriate keystrokes to *MSTS*. This "servo loop" approach to control tracking is delineated in the Interface Functional Block Diagram inset box.

It was discovered that thanks to the Windows XP operating system, communication from the *MSTS* patches to COM1 had to employ a combination of *HyperTerminal* (under Accessories / Communications) and UserPort (Link 6 and Link 7). Windows does try to protect its resources!

One other noteworthy aspect that required consideration in the interface design was the variation in *MSTS* frame rate during any given operating session, which is typically more than 3:1 depending on the degree of external visual detail and view selection. This variation is accommodated by accessing the frame rate parameter within *MSTS* and dynamically adjusting the output rate in the software patch by skipping cycles as the frame rate increases. This allows the Stamp microcontroller to keep up with the outputs and achieve a relatively constant refresh rate (approximately 10/second) for driving gauges and reading controls. As a side note, the application of the software patch, which constitutes a relatively small amount of code in comparison to *MSTS* itself, was found to have no detectable impact on frame rate.

Experiences in the Cab

Simply stated: **WOW**, there's nothing like it (short of the real thing)!! I originally thought *MSTS* was pretty neat while operating from a PC keyboard. Then, I discovered a new experience using *RailDriver*. Now, operating from a cab brings a quantum leap in feeling like you're part of the simulated machine and environment. I'll highlight three specific things that got my attention.

First, it was immediately obvious that operating a steam train is truly a three person job. When an engineer has to leave the seat to open the fire door or adjust the fire controls, having a fireman is very desirable. Or, leaving the cab as a brakeman to throw a turnout, set a handbrake, or open a coupler leaves nobody to control the locomotive. Physically having to do all these things versus having the functions within finger length on a keyboard is a much different experience. I had originally considered having a fireman's jump seat in the cab – I now believe it's a must-do addition. I did have sufficient foresight to add the brakeman's station outside the cab. Now, I need two more people as obsessed as I am about train simulation to be on the MUTT crew.

The second thing that became apparent is how little I had absorbed from the various prototype documents dealing with braking, fire control, and even basic "set up and running" functions of the engineer. More than once, while concentrating on some driving operation, I neglected to watch the water sight glass and apply the injector controls, resulting in melted boiler plugs (and a mandatory *MSTS* activity restart). In retrospect, while I had used various prototype documents for simulator hardware design I had not sufficiently read and digested them from an operations viewpoint.

The third aspect is that I gained a much better perception of the locomotive physics, and have begun to question certain aspects having to do primarily with the power factors and braking characteristics. The locomotive model is not any different than with keyboard operation, but sitting in the cab feeling the sound system vibration, watching the gauges, and monitoring the response to control changes gives one a different perspective on the physics model. This is another area where I have some homework to do for the specific steam locomotives I am operating.

In short, the cab experience is sufficiently real and I think all that is missing are uncomfortable heat, steam leaking from fittings, wind coming in through the front window (skip rain), and more vibration and bumping in the seat. I'm now working on a design change for the latter effect – just like a model railroad layout, MUTT probably never will be completely finished.

The Future of "P-Scale" Virtual Railroading

One aspect I have saved for last is that the method of using CE to locate the source of variables for output and the overall simulator architecture are not unique to *MSTS* or steam cabs. The approach could be implemented for any of the other train simulation software platforms (e.g., *Trainz* or *Rail Simulator*), any other type of constructed cab (e.g., diesel or electric), or for that matter to simulations other than trains. Therefore, the simulator design I have described can be applied to a diverse community interest – essentially any game platform that is controlled with keyboard inputs is a candidate.

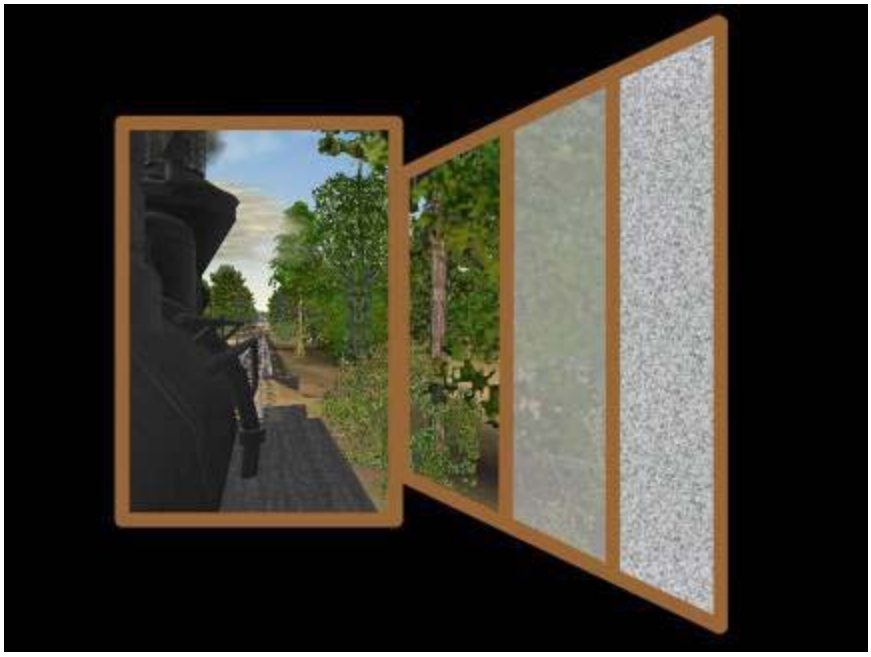
Throughout the project I reflected on how MUTT compared to my prior model railroad layouts. A conventional layout, in any scale G to Z and all others in between, is a simulation of the real world. I have coined the term "P-scale", or Prototype Scale, to represent a class of simulation that employs a full size cab with "human in the loop" operation. It is an extension of classical virtual railroading in the use of computer generated dynamic and visual models of railroad elements with the added dimension of prototypical cab hardware controls and displays. With this definition in mind, consider the following comparison of P-scale virtual railroading to model railroad layouts:

Feature	P-scale Virtual RR	Model RR Layout
Physical space	Fixed footprint	Grows to fill available space
Skills - hardware	- Carpentry/woodworking - Electrical/electronics - Mechanical/metalworking - Painting/artistic media	(requires the same skill set)
Skills - modeling	Software objects and effects: - Download - Scratch build (3D) - Purchase from third parties - Incorporate lighting effects - Include sound files	Physical objects and effects: - Kit assembly - Scratch build - Purchase finished - Special lighting electronics - Special sound electronics
Expense - hardware	(depends on design features)	(roughly the same)
Expense - models	- Extensive freeware available - No cost to scratch build	- Sizable investment over time - Materials for scratch building
Configuration flexibility	- Fixed locomotive type - Any place, era, time, or season - Extensive use of animation	- Wide variety of locomotives - Fixed location, era, and season - Special hardware to animate
Operating sessions	- Multiple operators - Other "AI" trains - Multi-player via internet	- Multiple operators - Multiple trains - Multi-player (as space permits)

The bottom line is that the primary limitations of P-scale versus a conventional layout are: (1) having a single locomotive cab configuration, and (2) the need to master certain software skills. In all other categories the advantages of P-scale in effect make it the equivalent of an endless set of layouts, each with it own historical location and seasons. And, rather than being an observer/operator watching and controlling from a distance, with P-scale one becomes an integral element of the simulation itself as an active participant performing in the roles of the crew.

My personal belief is that P-scale has the potential to become as popular a hobby in the future as model railroading has been in the past. I do not believe it will (or should) replace model railroad layouts – there is a certain magic of watching trains runs through miniature scenes that is rooted in childhood memories, at least those of us in the older age group. But today's young generation are raised on computer games and may likely find virtual railroading more in line with their interests and skills. P-scale offers something for both interest groups – the challenges of building something with tools and physical effort, very similar to conventional layouts, and the added challenges of applying computer skills to build something with bits and bytes. If third parties decide to pursue the market potential, similar to what *RailDriver* has done and what others produce for flight simulation, we might witness the birth of a software and hardware product line that could rival the HO-Scale model train industry's 50+year success.

End of editorial – time will tell if my vision is correct. In the meantime, I intend to resume modeling a local historical branch line and spend a lot of time in the cab. With some practice, I might eventually be worthy of "set up and running" engineer status.



Questions and comments regarding the MUTT project can best be communicated to me via the Train-Sim.Com forum (Link 8) under Microsoft Train Simulator. The forum also provides an option for contact via private email.