ACLM

REFERENCE MANUAL

ACLIENT APPLICATION

Sacramento Municipal Utility District
Telecommunications Department
Engineering Design Group

Revision 3

November 19, 2004

CONTENTS

CONTENTS

CONTENTS

CONTENTS

INTRODUCTION

Purpose

Manual

This manual provides user and technical information for the ACLIENT application used with the Air Conditioning Load Management (ACLM) System. Other information about this application and the ACLM System is contained in the *ACLM System Reference Manual*.

ACLIENT Application

The ACLIENT application allows authorized District employees to temporarily deactivate load controllers attached to customer air conditioners. These deactivations are normally performed by the Customer Services Department's Operations Support Group in response to customer requests for temporary relief from elevated temperatures resulting from cycling operations.

The ACLIENT application also allows users to remotely perform other ACLM functions such as activating/deactivating safety lights, viewing data base reports and monitoring cycling status.

Terminology

Organizations

The term "District" refers to the Sacramento Municipal Utility District (SMUD). The term "Telecommunications" refers to the District's Telecommunications Division. The terms "Design" and "Maintenance" refer to the Engineering Design and Maintenance groups within Telecommunications, respectively.

Software

The term "software" is used to denote programs and data residing on a PC, peripheral or microcontroller.

The term "application" denotes the executable software program or script and all of the auxiliary files, support programs, and configuration data needed for proper functioning. This term also applies to the output files that the program produces, including log and data exchange files.

Excluded from the application are "support software" such as operating systems, compilers and diagnostic routines.

The term "program" generally refers to an executable program or the source code and modules needed to compile it into a standalone executable.

Hardware

The term "hardware" refers to the physical equipment, firmware, connections and configuration needed to support system functions, particularly application and support software.

System

The term "system" refers to a combination of hardware, software and applications performing a specific function. The term "subsystem" refers to systems that perform functions that are integrated into a higher-level system or subsystem.

Overview

Approach

This manual describes the usage, design and operation of the ACLIENT application.

Because of the relative simplicity of this application, there are no separate user, technical and operation manuals. The manual has separate chapters dedicated to these functions so that the readers do not have to extract information from several different parts of the manual. Because of this approach, some information may be duplicated.

Assumptions

It is assumed that all readers are generally familiar with the purpose and functioning of the ACLM System and with common terms used to describe its components.

Readers not familiar with the District's ACLM program and its operation should refer to the *ACLM System Reference Manual*.

USER GUIDE

Overview

Purpose

The ACLM Program reduces overall District power load on hot summer days when there are power shortages or other power system emergencies. The power load is reduced using radio-controlled load switches (switches) to periodically turn off air conditioners serving commercial and residential customers.

During cycling operations some customers may experience emergencies related to the elevated temperatures that cycling can cause. To respond to these emergency situations, authorized District employees can temporarily disable individual switches. The ACLIENT application (ACLIENT) allows users to perform this function and other functions, as described in following sections.

The acronym ACLIENT is a shortened version of "ACLM Client."

Switch Deactivation

The deactivations performed by ACLIENT are temporary.

Normally, changes to customer switch programming are performed by an SAP application. Once a day, the SAP application sends an update file to the ACLM System. The update file contains information about the customer option changes performed during the day; the ACLM System uses this SAP extract information to program the customer load switches.

The only "official" changes to customer switch programming are via the SAP application and the daily extract file. To ensure that the database maintained by the ACLM System agrees with the SAP database, SAP periodically generates a synchronization file containing all of the switches in the SAP database. The ACLM System uses this file to reprogram any switches in its database that do not agree with the switch information in the SAP synchronization file.

Thus, any temporary switch disabling (initialization) performed with ACLIENT will be reversed when the SAP and the ACLM System synchronize their databases: to make ACLIENT switch initializations permanent, the changes must be entered into SAP.

Safety Light Lights

The ACLIENT application allows the Safety Lights at the 59th Street Yard to be activated or deactivated remotely from any Internet browser, such as Internet Explorer, on the District's CORPORATE network. This process is similar to that used to perform temporary switch deactivations (i.e., access is via the ACLIENT home page and information is entered in an HTML form).

This capability supersedes the old method of remote activation and deactivation using the LIDA remote client software. The LIDA Client is now obsolete and the executable, LIDA.EXE, can be deleted from user PCs.

The following sections describe how to use the browser controls.

Note that the Safety Lights can still be manually activated or deactivated at the Master Station PC using a menu option. Also, the Safety Lights will automatically deactivate after 24 hours of operation, regardless of the activation method.

Access

Browser Address

To access the ACLIENT application, enter "ACLM1" in the "Address" data entry field on the browser of any PC connected to the District's CORPORATE domain. Typically, the browser used will be Internet Explorer.

Note that the ACLIENT application cannot be accessed via the Internet except via District-authorized remote clients. Normally, such remote deactivations are both unnecessary and undesirable.

Figure 1 shows a browser screen with the correct ACLIENT access entry.

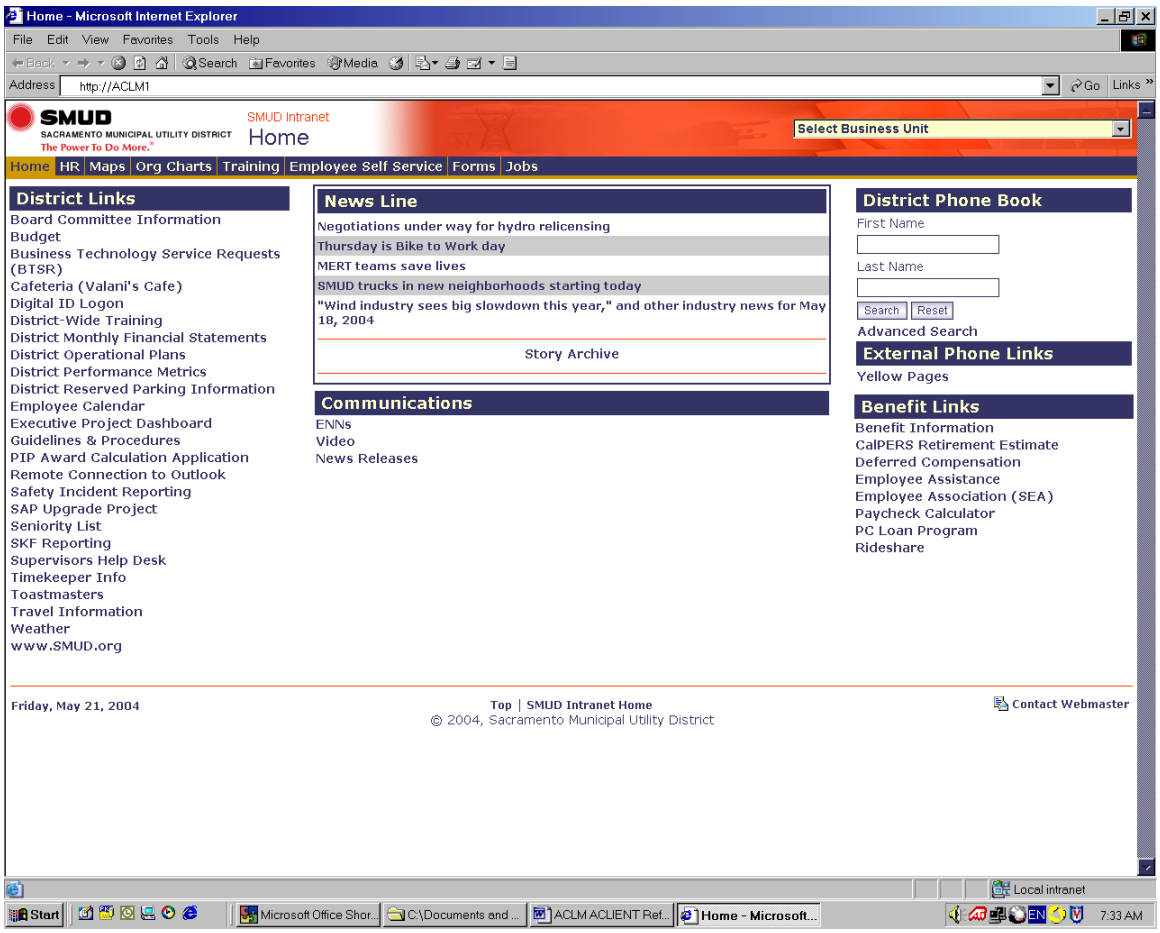

Figure 1. **ACLIENT Browser Access**

Home Page

Figure 2 shows the ACLIENT Home Page resulting from the selection in Figure 1.

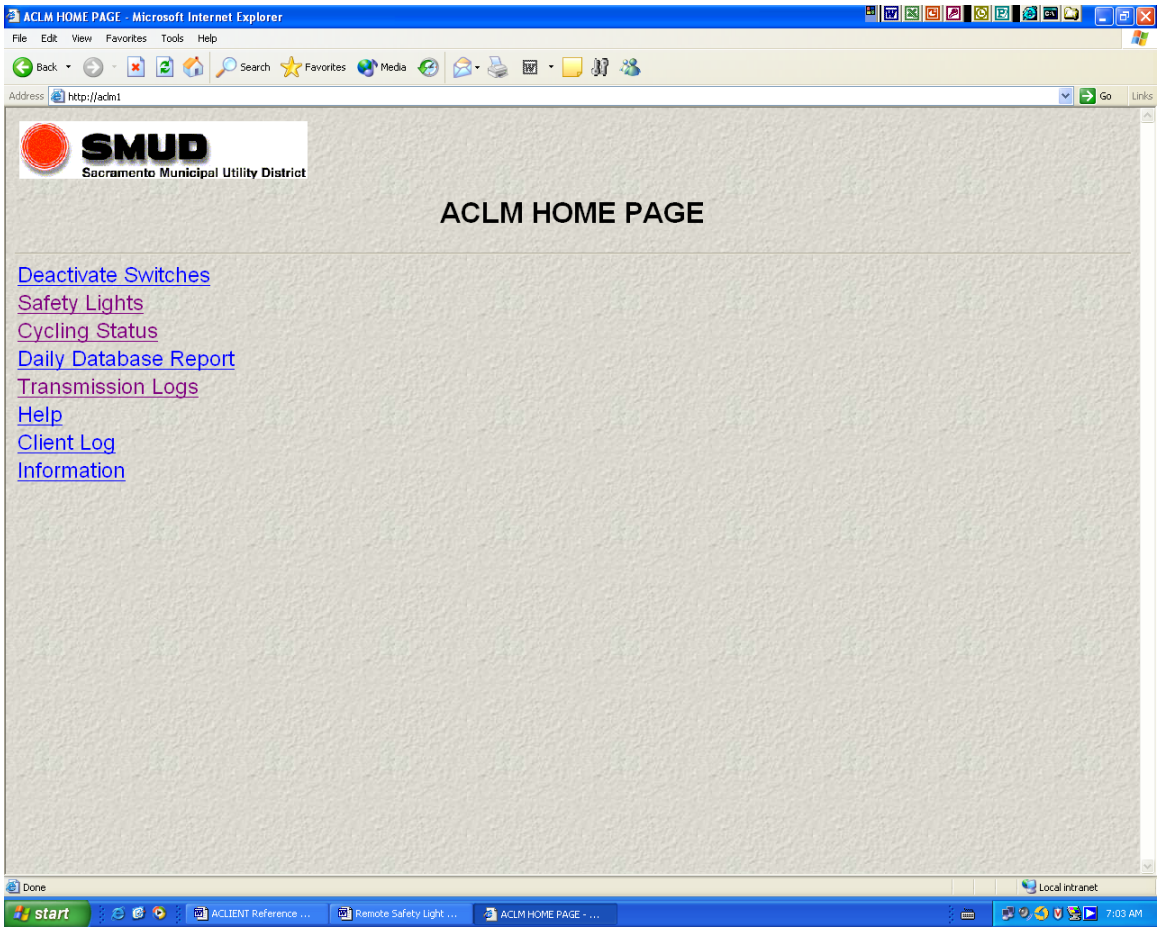

Figure 2. **ACLIENT Browser Access – ACLM Home Page**

To perform deactivations, select the "Deactivation Form" option. Figure 3 shows the resulting page. The other options will be described later.

Deactivate Switches

Figure 3 shows the ACLIENT Deactivation Form, including typical data in each field.

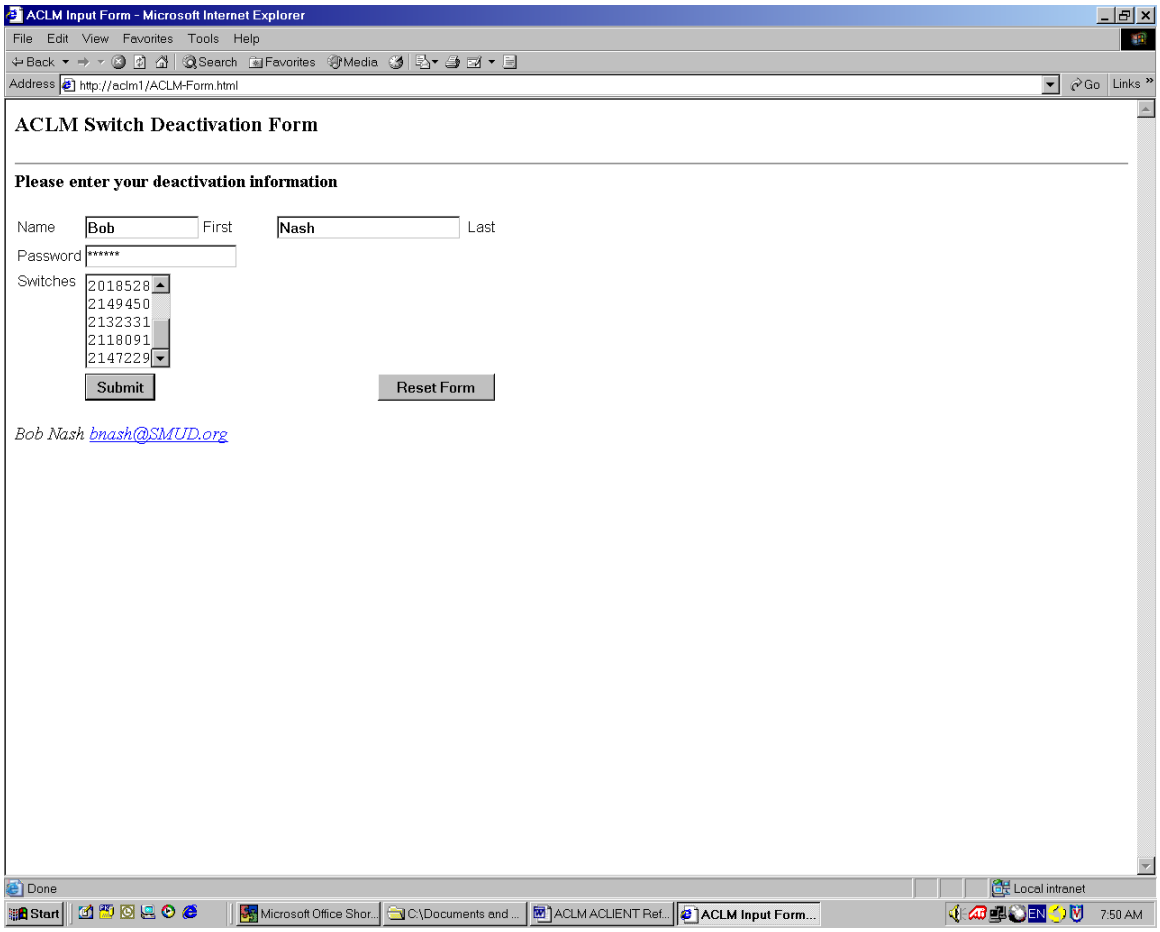

Figure 3. **Deactivation Form With Typical Data**

Name Entries

Users are requested to enter their first and last names. These are recorded in a client log on the ACLM Host controller to aid in troubleshooting and to provide a record of user activity.

The user name is also stored on the user's PC as a "cookie." Should the user fail to enter a user name, the user name from the user's cookie is entered in the client history log. The client log also contains the user's IP address so that entries can be tied to particular PCs.

Password

Users must enter a password to enable deactivations. If an incorrect password is entered, ACLIENT will display an error page. To correct an incorrect password, users can use the "Back" arrow at the top of the browser display to return to the deactivation form. Note that the user data is preserved so that the data does not have to be re-entered after a password error.

Deactivation Data Entry

The Deactivation Form provides a small text entry field for entry of switch (cycler) numbers. The data entry form is seven characters wide: this helps users visually validate the size of switch numbers. All switch numbers are seven digits long.

The data entry field does not provide any protection against entry of non-numeric or incorrect switch numbers: switch data is validated after it is submitted.

Switch data validation is as follows:

1. Switch numbers may be entered in a continuous sequence or they may be entered with one or more carriage returns between them,
2. Non-numeric data between switch entries is ignored (e.g., carriage returns, blanks, commas and periods),
3. Short switch numbers will be ignored, but will be shown as errors,
4. Non-numeric characters at the start and end of the switch entry field are ignored,
5. Switches outside the District's switch range of 20000000 to 2400000 are ignored but are shown as errors.

Status

After the Deactivation Form is complete, users can submit their deactivation candidates by clicking on the "Submit" button. The ACLIENT script then validates switch entries displays the results. Figure 4 shows a Deactivation Form with deliberate errors and Figure 5 shows the resulting validation page.

The validation page shows both the valid deactivation entries and the errors found during processing. Valid switches are displayed in blue and errors are displayed in red. The validation page also displays, in blue, the raw data entries.

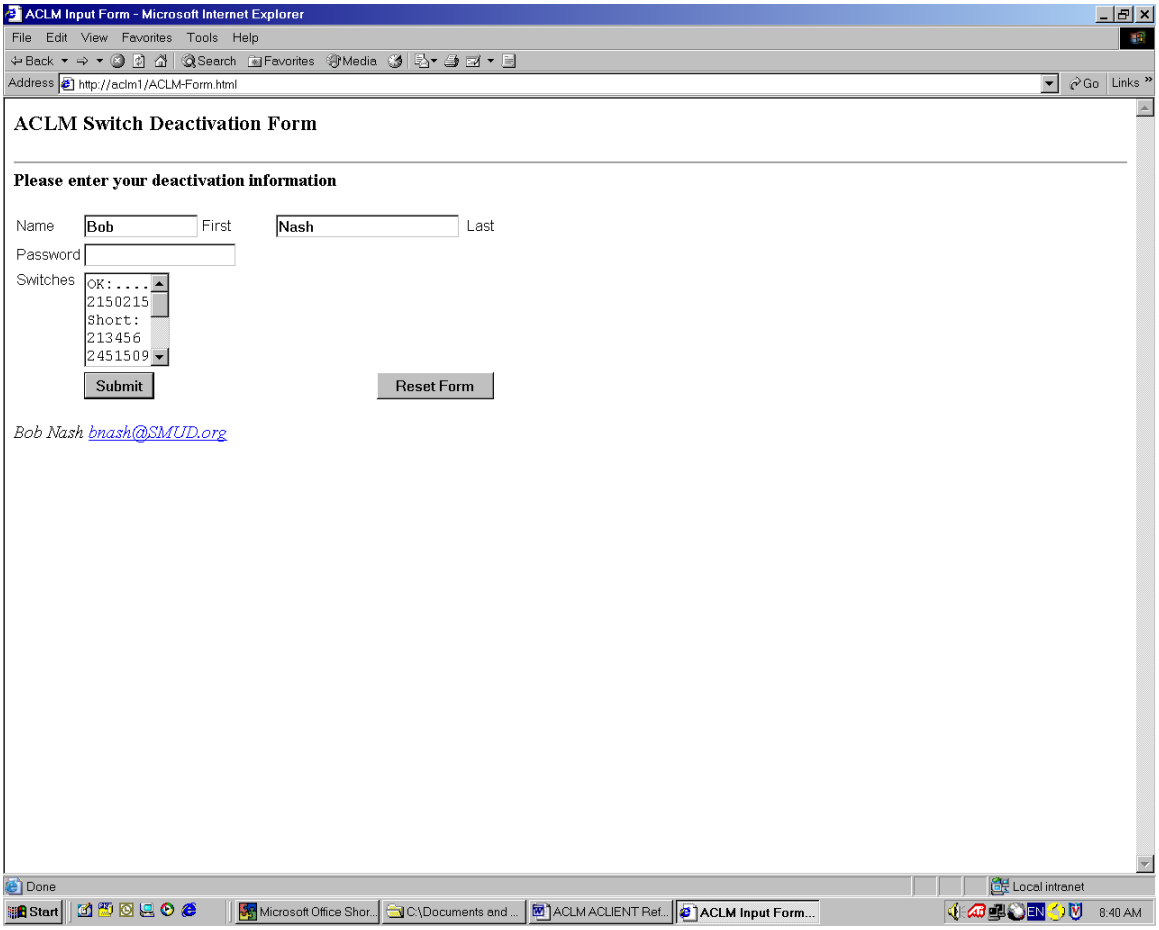

Figure 4. **Deactivation Form With Errors**

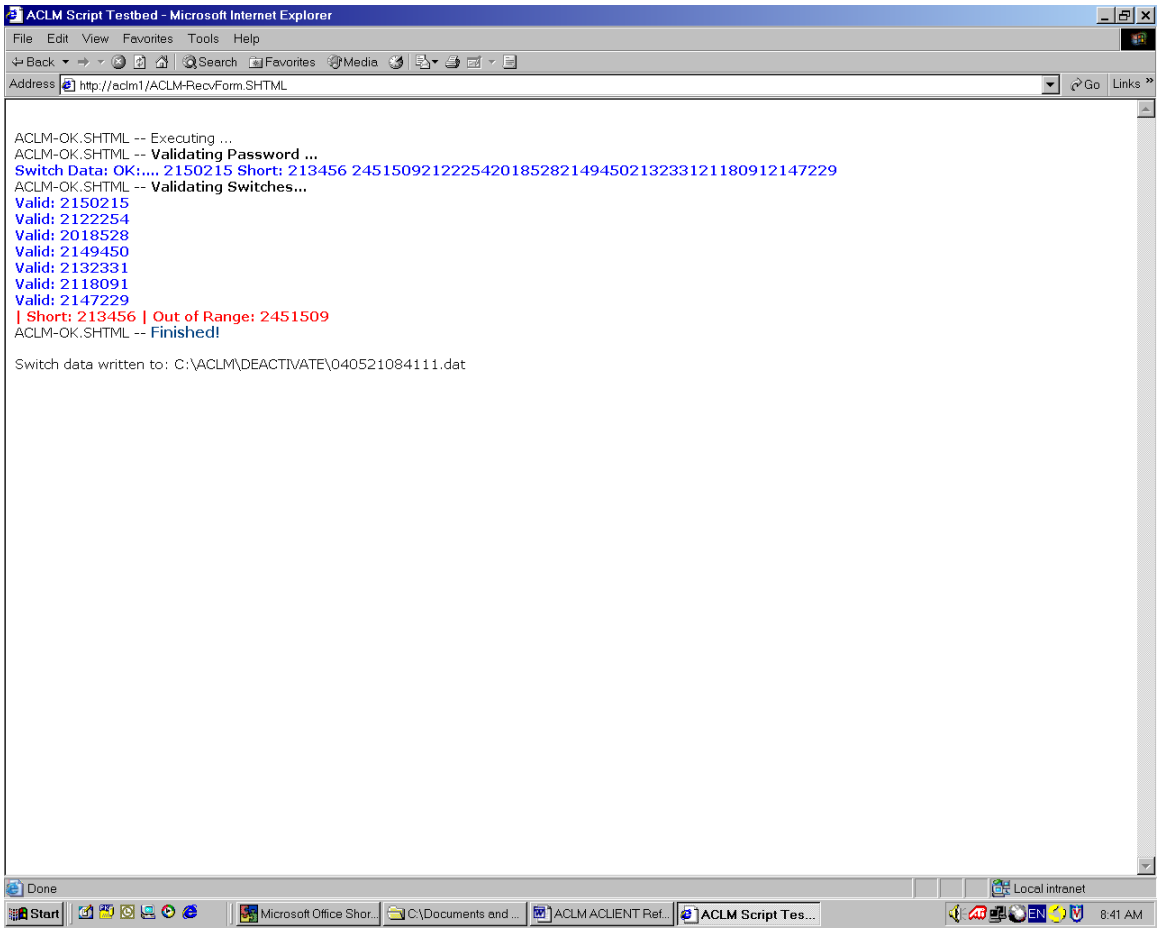

Figure 5. **Deactivation Status Page With Errors**

Also note that the Status Page shows the name of the switch deactivation file that was generated on the ACLM Host PC. This information is of little use to typical users, but can be useful during troubleshooting and system verification.

Duplicate entries are not treated as errors. Although they will produce duplicate initialization transmissions, these are mostly benign. However, duplicate entries do consume precious ACLM air time and may delay the deactivation of users with switch numbers entered after the duplicates.

Safety Lights

Activation

To activate the Safety Lights, enter your first and last name, the Safety Light password and the requested action: for activation, enter ”start” in the Action field, as shown in Figure 2 above.

Although the password is case sensitive, the action is not. Leading and trailing blanks in the action field are also ignored.

Note: if there are any items already filled in when the Safety Lights panel is displayed, it is best to select the ”Reset Form” button to clear out all previously entered data.

When the name, password and start action are entered, click on the ”Submit” button to activate the Safety Lights.

Deactivation

To deactivate the Safety Lights, follow the instructions above for activation, but enter ”stop” in the Action field. Similar to the “start” action, the stop action is not case sensitive and may contain leading or trailing blanks.

Status

After submitting a ”start” or ”stop” action, the browser will display a status screen. Figure 6 shows a typical status panel for a valid action entry; Figure 7 shows a typical status panel for an invalid action entry.

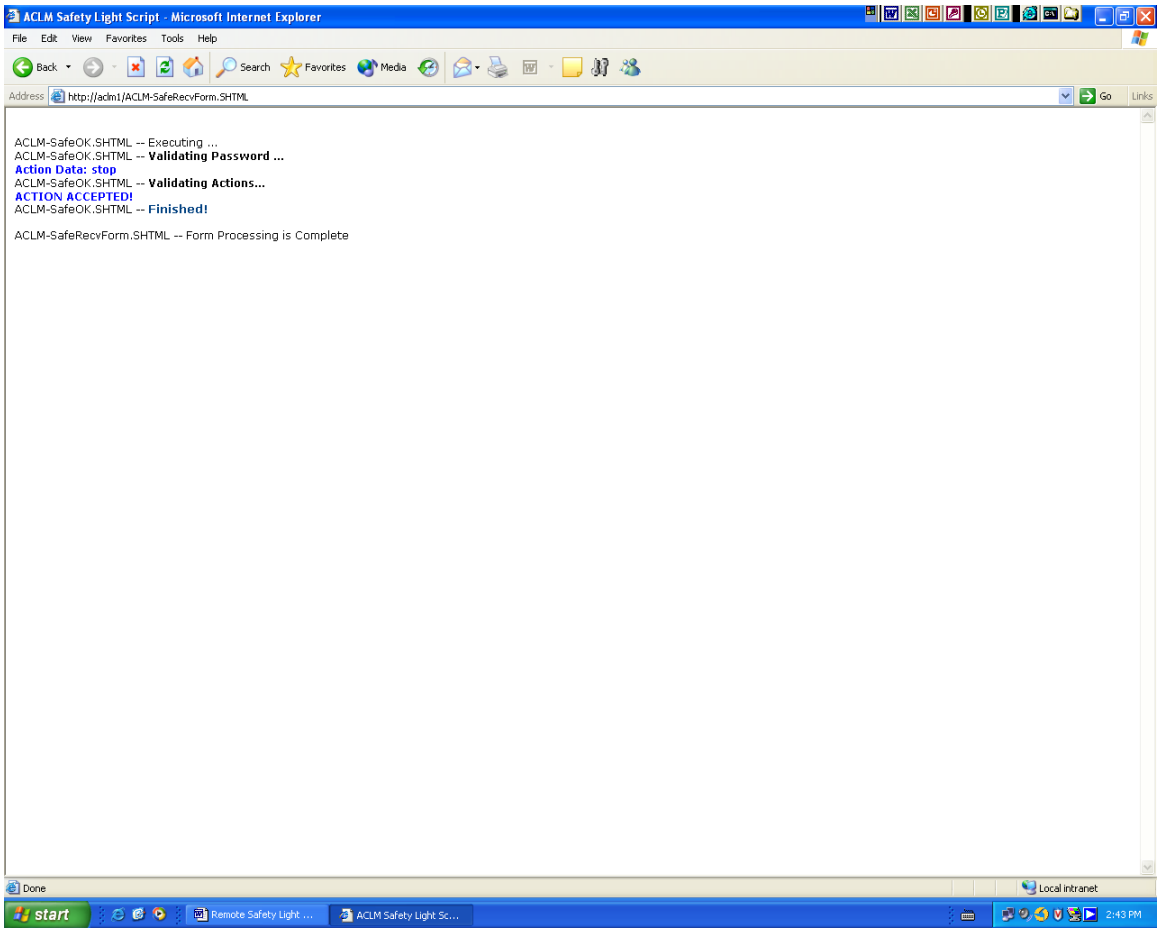

Figure 6. **Safety Light Status Display – Valid Action**

Status (Cont.)

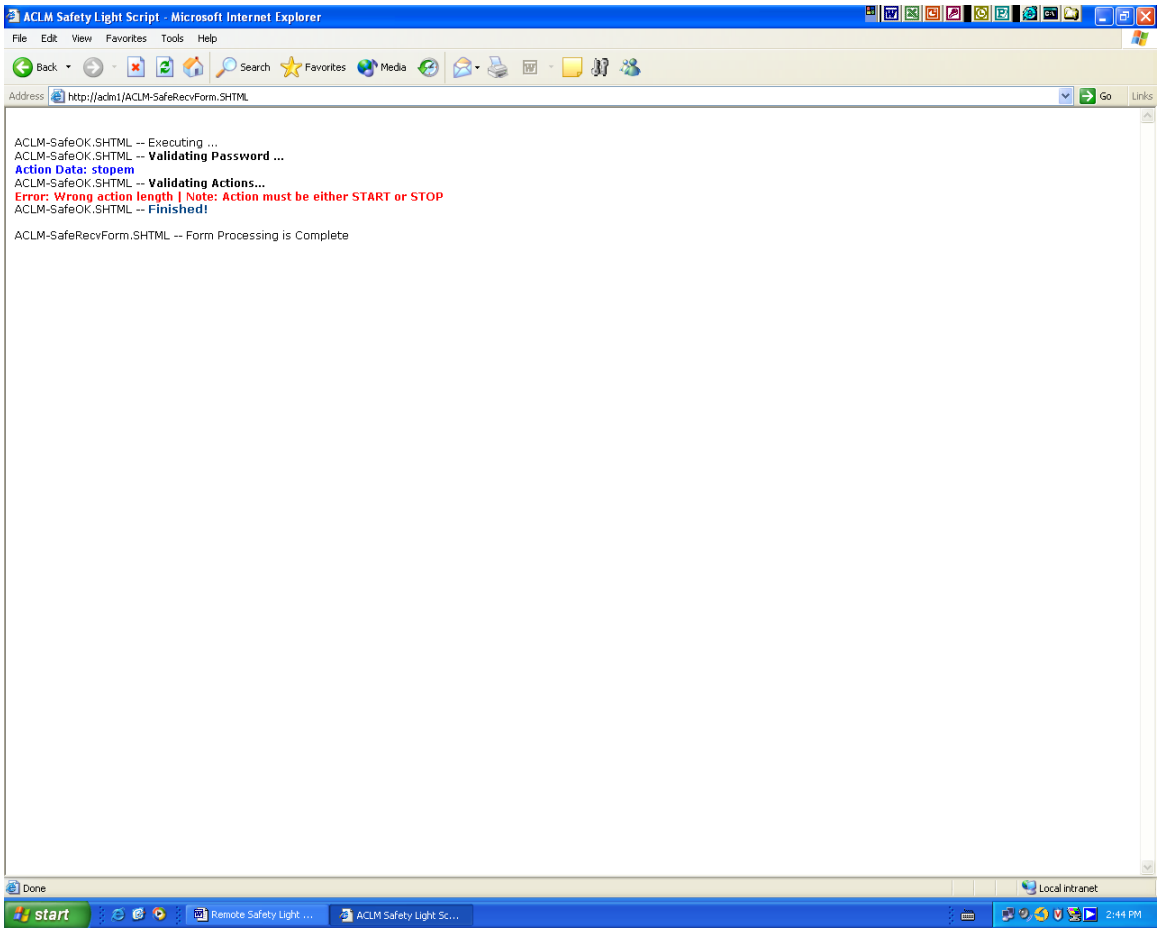

Figure 7. **Safety Light Status Display – Invalid Action Entry**

Confirmation

To confirm that the Safety Lights have been activated or deactivated, go to the ACLM Home Page and select the "Cycling Status" option. This option is described in more detail in a following section.

To confirm that the Master Station has received activation or deactivation commands, observe the Safety Light activation status at the bottom of the Cycling Status display.

Figure 5 shows a cycling status panel with an "ACTIVATED!" status for the Safety Lights; Figure 6 shows a panel with a "DEACTIVATED!" status for the Safety Lights.

Note: The start and stop actions are checked only once a minute and the Safety Light activation commands are sent only every three minutes. Thus, it may be a minute before the Cycling Status shows "Activated!" and up to four minutes before the lights start flashing.

Cycling Status

The Cycling Status option is accessed from the ACLM Home Page, as shown in Figure 2 above. After the option is selected, a panel similar to the ones shown in Figures 8 and 9 will be displayed.

As shown, the status panel gives the current ACLM time and the megawatt and temperature values from the Master Station. Also shown is the cycling status: NORMAL indicates no cycling and CYCLING! indicates that cycling is in progress. When cycling is in progress, the cycling level is also displayed. For normal operation, the indication is "0% Res 0% Com."

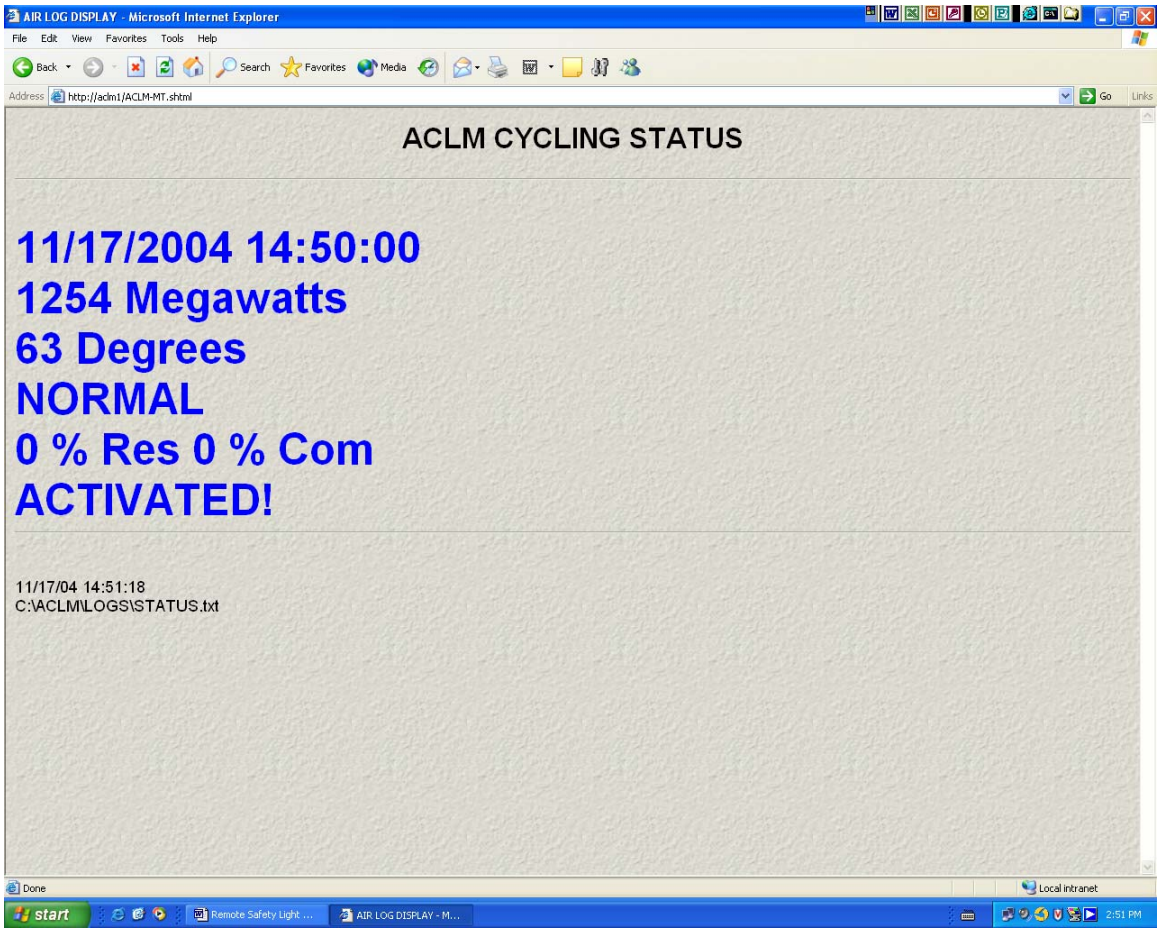

Figure 8. **Cycling Status Display – Activated Safety Lights**

Cycling Status (Cont.)

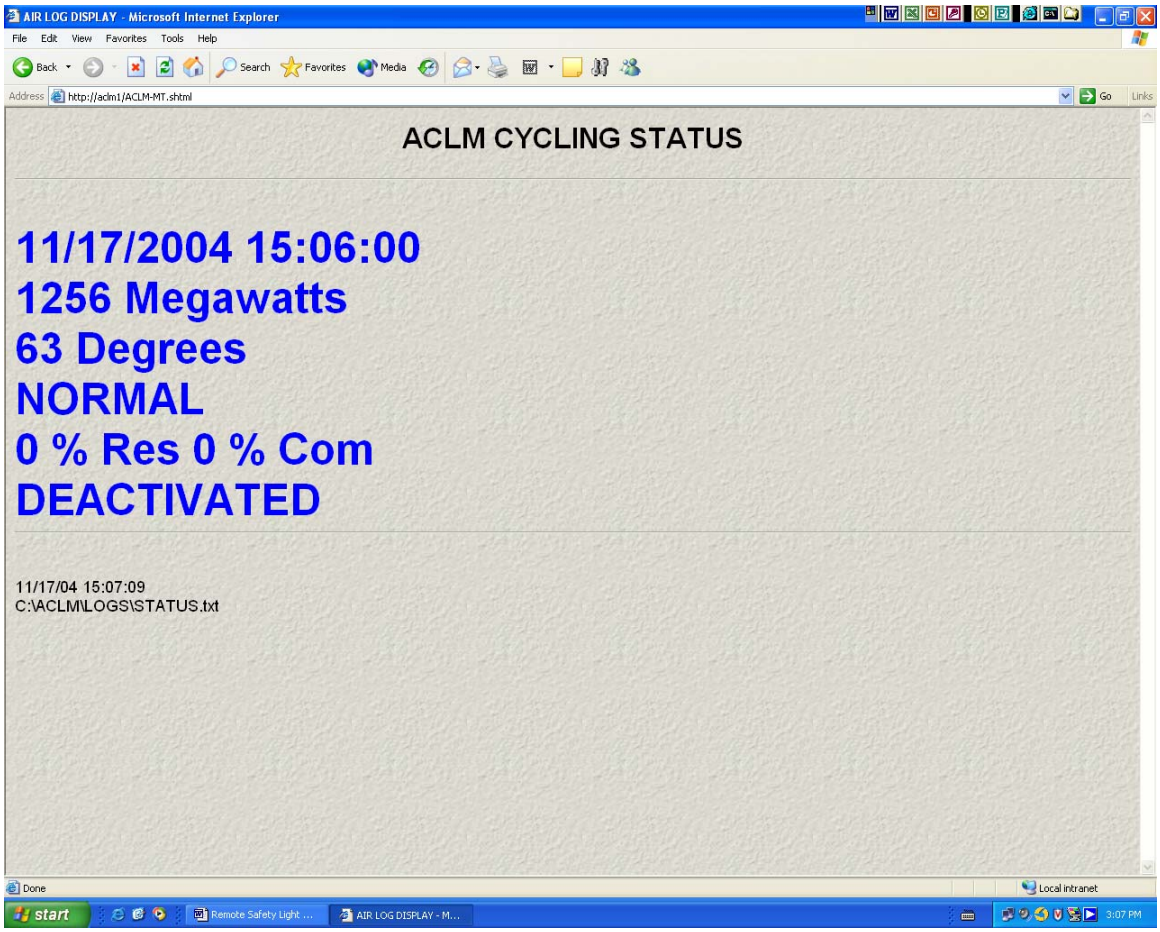

Figure 9. **Cycling Status Display – Deactivated Safety Lights**

Daily Database Report

The Database Report option is accessed from the ACLM Home Page, as shown in Figure 2 above. After the option is selected, a panel similar to the one shown in Figure 10 will be displayed.

Header

As shown, the report header includes the "printed" date, the format of the data file (e.g., delimited data format) and a list of invalid switches encountered during report processing.

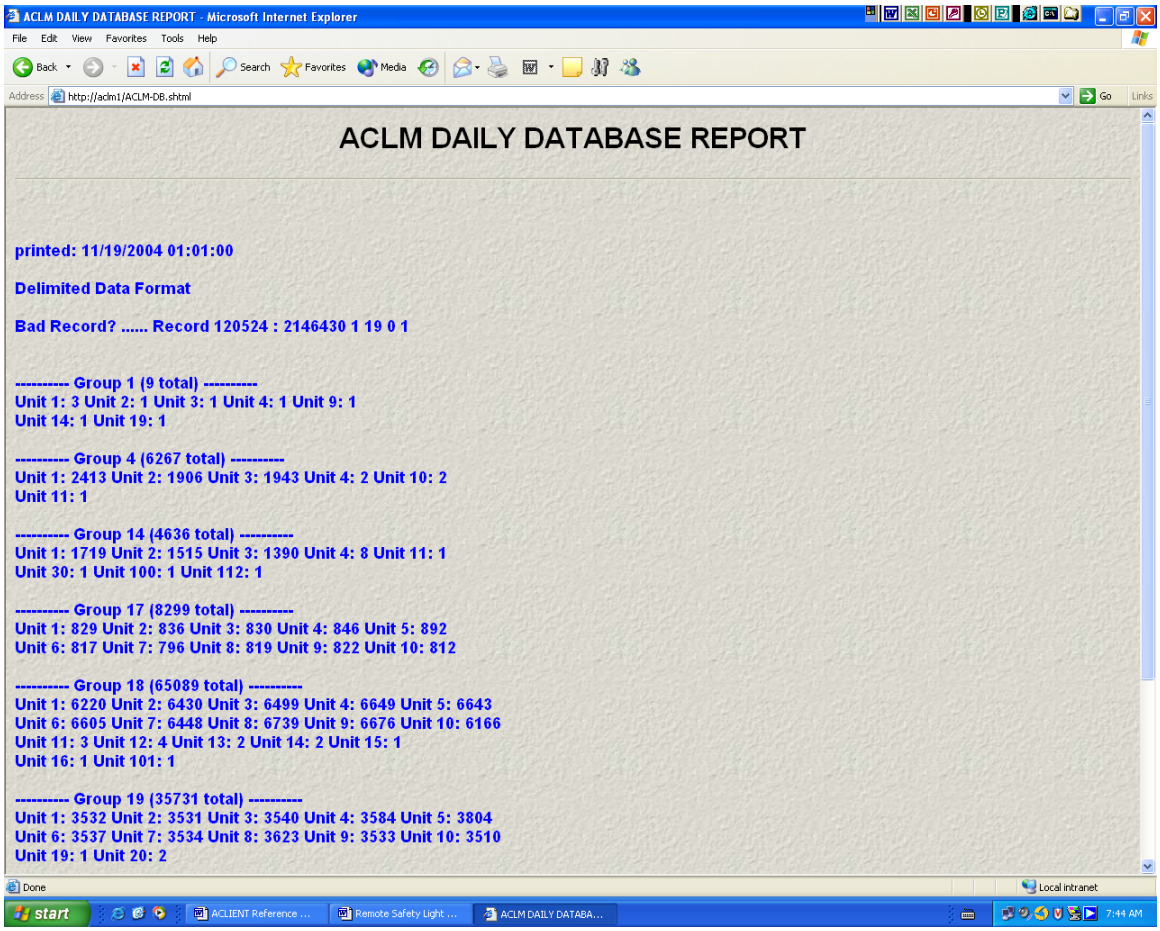

Figure 10. **Daily Database Report With Header Information**

Report Body

The body of the Database Report provides a group by group and unit summary of the switches in the ACLM database. Because the report is produced early in the day, the report information is typically very accurate: SAP database synchronizations and switch downloads are performed late in the day and are typically complete by the end of the day.

Report Summary

As shown in Figure 11, the end of the Database Report provides a summary of the switches in the ACLM database, including the total number of switches, the number of active switches, the number of initialized switches and the number of invalid switches (usually data entry errors for unused group or unit numbers).

As for most other ACLIENT reports, there is a footer that shows the text file that is the source for the Database Report (e.g., C:\ACLM\DAILY-REPORT.txt) and the date and time that the report was displayed by the user.

The "EVAL FORGET-SCRIPT" text at the bottom of the display can be ignored. It is used to confirm that the Database Report script processing was terminated correctly.

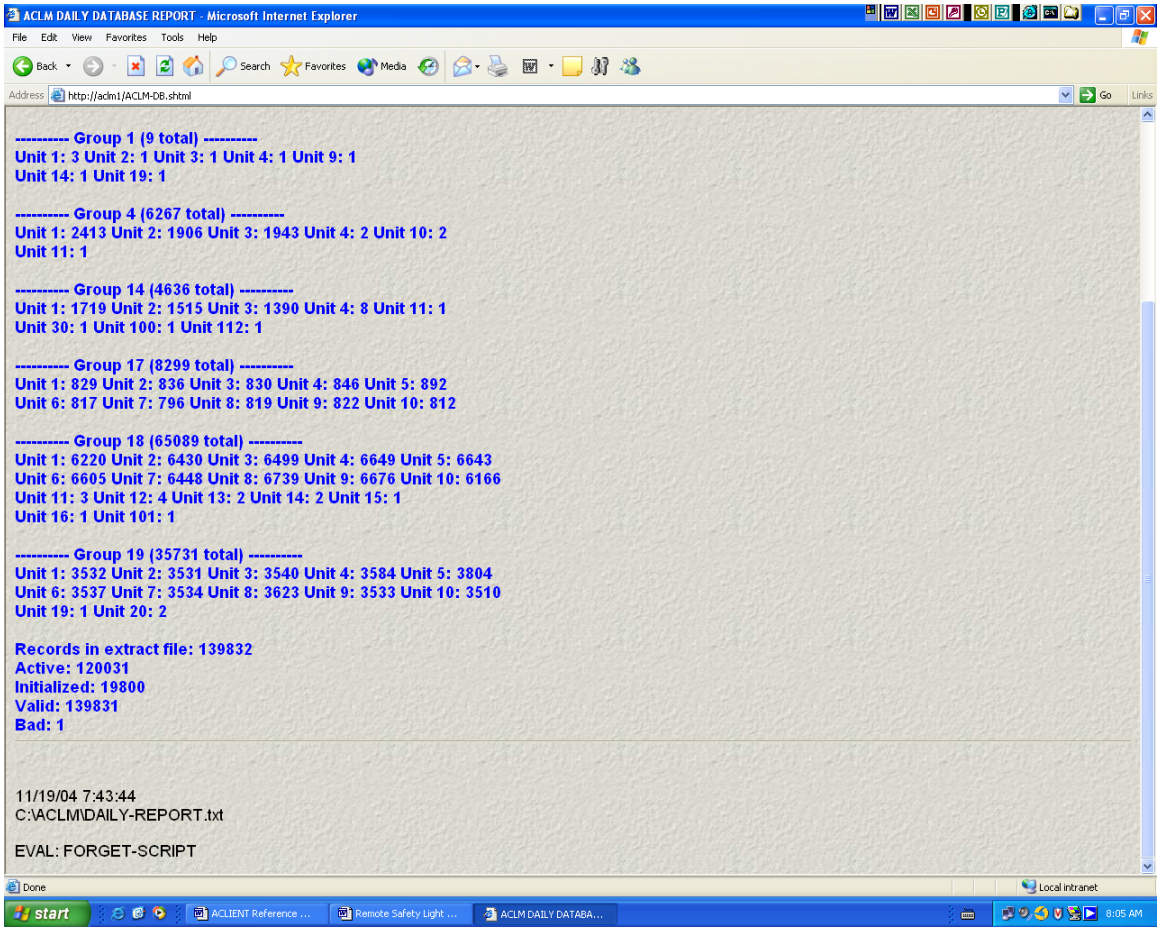

Figure 11. **Daily Database Report With Footer Information**

Transmission Log

The Transmission Logs option is accessed from the ACLM Home Page, as shown in Figure 2 above. After the option is selected, a panel similar to the one shown in Figure 12 will be displayed. The panel provides options for displaying the transmission log for the current day or the transmission log for the previous day.

Log Display

As shown in Figure 14, the Transmission Log display shows each transmission sent during the day and the time that it was sent. Transmissions are shown in both numerically and mnemonically coded format. The mnemonics used for the log are displayed at the start of the log, as shown in Figure 13.

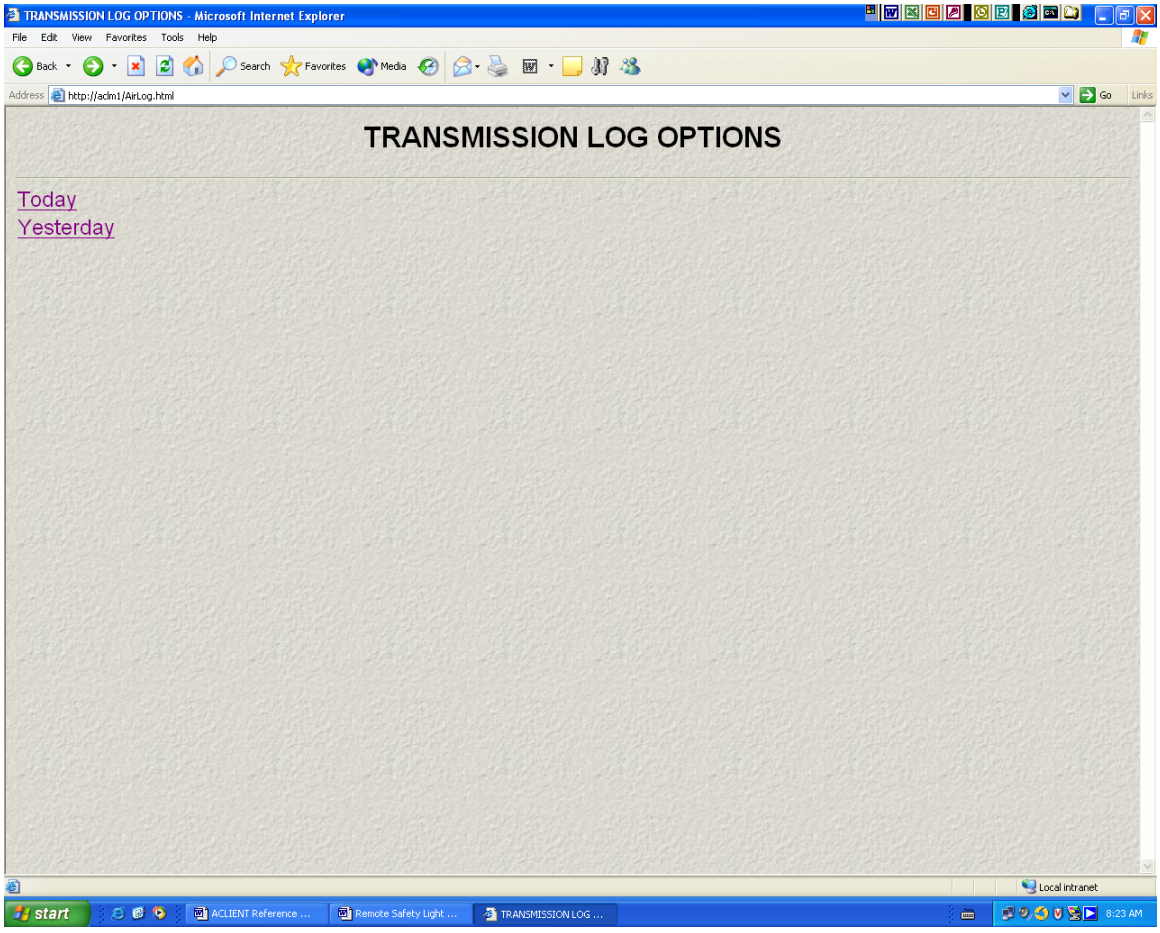

Figure 12. **Transmission Log Options**

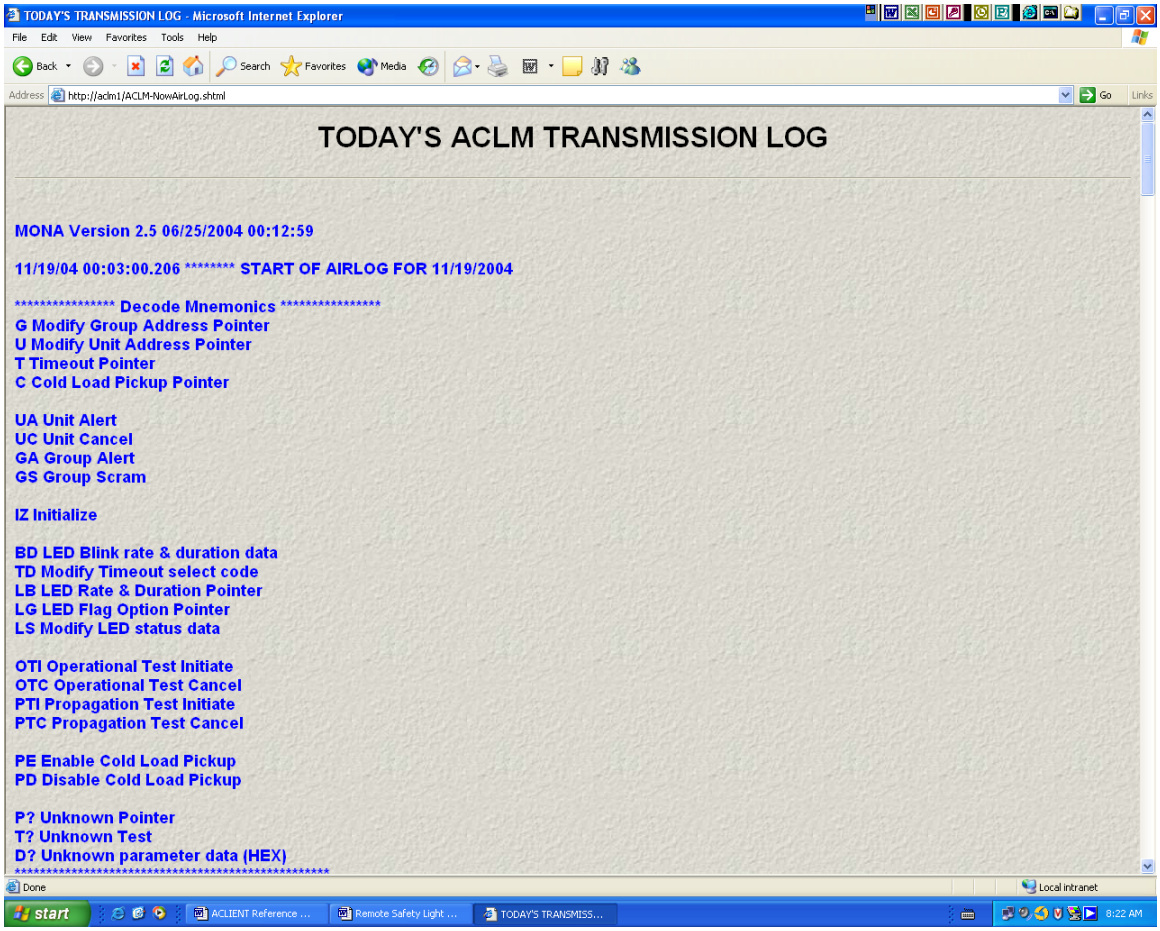

Figure 13. **Transmission Log Header**

Header

The Transmission Log is produced by the MONA off-air transmission monitoring program residing on the ACLM HOST computer. As Figure 13 shows, the Transmission Log header includes the version of the MONA program used to generate the log file, the date and time that the log file was started and a list of the mnemonics assigned to various command codes.

Body

As shown in Figure 14, the most common transmission type in the body of the Transmission Log is the "watchdog" signal that is sent once in every three minute transmission window. Note that numeric code for this (in hexadecimal) is "EF8" and the assigned mnemonic is "PTC" for "Propagation Test Cancel."

Also note that the numeric code is given in hexadecimal to facilitate conversion to the binary code that represents the actual bit pattern sent by the transmitter. For commands that take more than one command transmission, such as switch programming sequences, the log may display several hexadecimal codes.

Figure 14 shows one of these types of transmissions for a switch being deactivated using the Deactivate Switches option described above. The multiple numeric command sequence of "906 A3C D8F EF7" indicates that multiple transmissions were required to initialize the switch. The actual switch number is embedded in this numeric sequence, along with the transmission type (e.g., "EF7" for initialize). The mnemonically decoded transmission is given immediately below the numeric sequence. In this case, the transmission is an initialization (IZ) for switch number 2150215 (a test switch).

Figure 15 shows a previous days' log with switch programming transmissions. Note the time of the transmissions. These are transmissions resulting from the switch programming file downloaded from SAP at approximately 22:00 (10 P.M.) each day. Displaying the log for this time period is a good way to verify that the SAP updates are being performed.

Note that the transmissions for programming a switch are much longer than for initialization and that the mnemonic decoding is also more detailed. The mnemonic decoding includes the switch number, the register number, the group number, the unit number and the timeout value. Because the Transmission Log is a direct record of the transmissions received off the air from the ACLM Master Station, it is the primary source to verify transmissions.

Footer

Similar to other ACLM displays, the footer of the Transmission Log contains the date that the log was viewed by the user and the name of the file that is the source of the display (e.g., C:\ACLM\LOGS\041119MA.log). The numeric part of the file name is the current date, in year/month/day sequence, followed by the characters "MA" to indicate that this is a MONA Air log. Other ACLM logs have the same date format, but different characters to indicate the log type.

The "EVAL FORGET-SCRIPT" text at the bottom of the display can be ignored. It is used to confirm that the Transmission Log display was processed and terminated correctly.

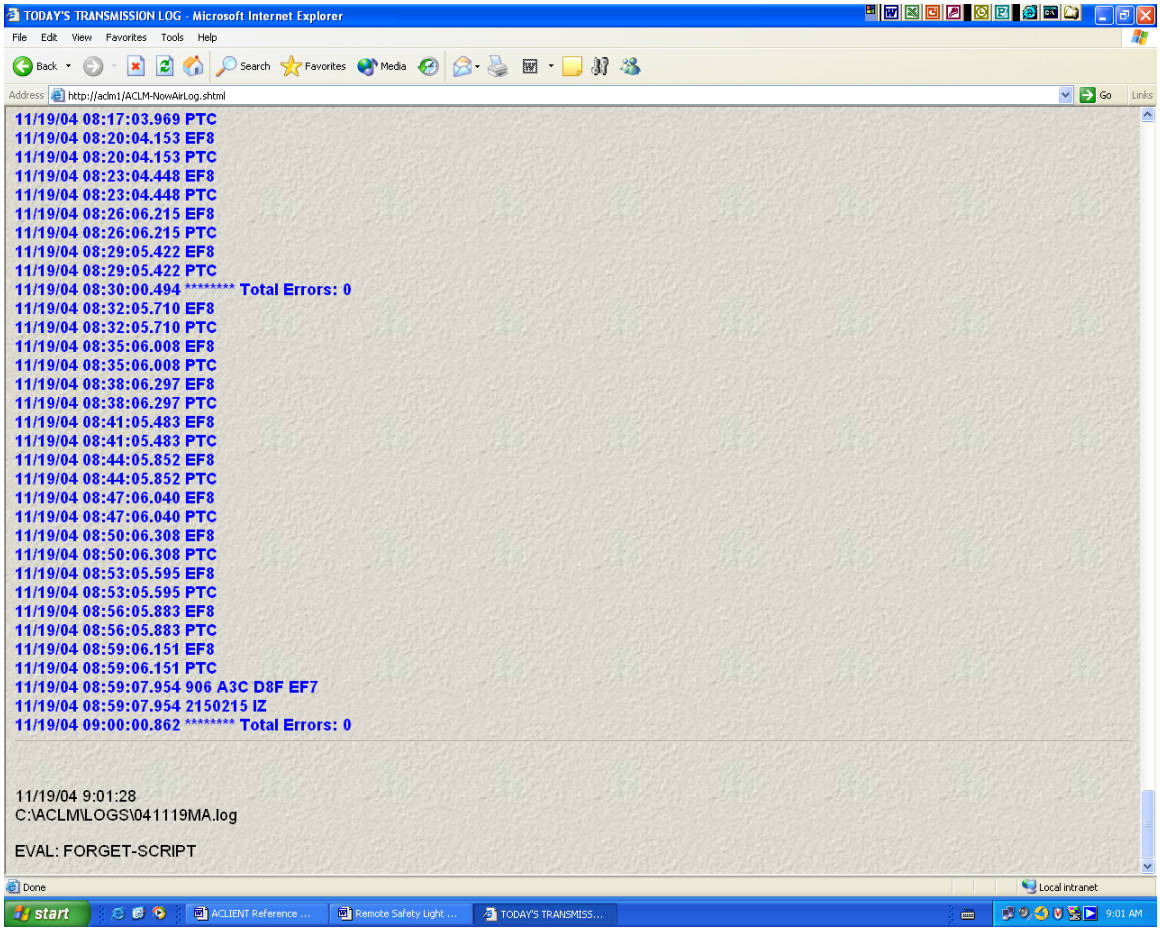

Figure 14. **Transmission Log With Footer Information**

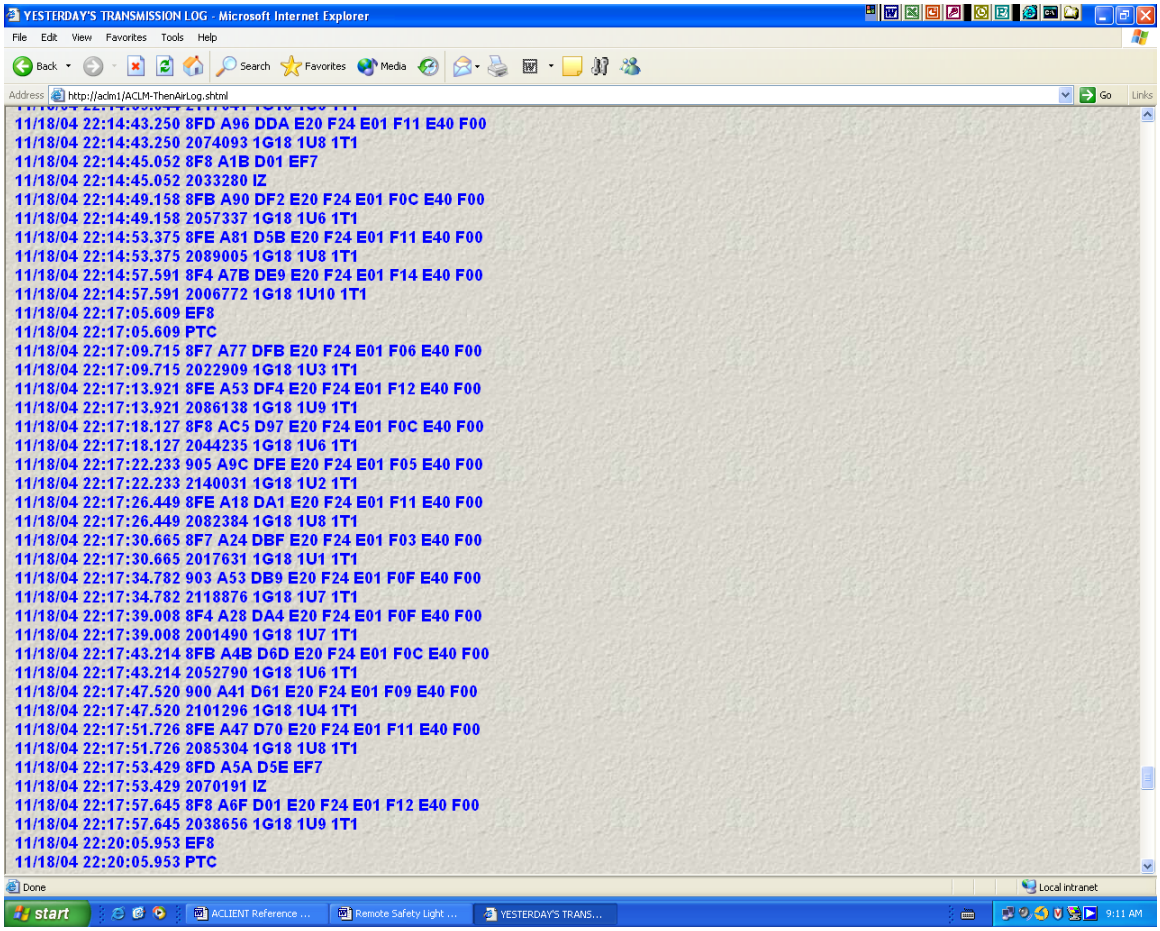

Figure 15. **Transmission Log Showing Switch Programming**

Help

The ACLM Help option is accessed from the ACLM Home Page. Figure 16 shows the first part of the Help panel.

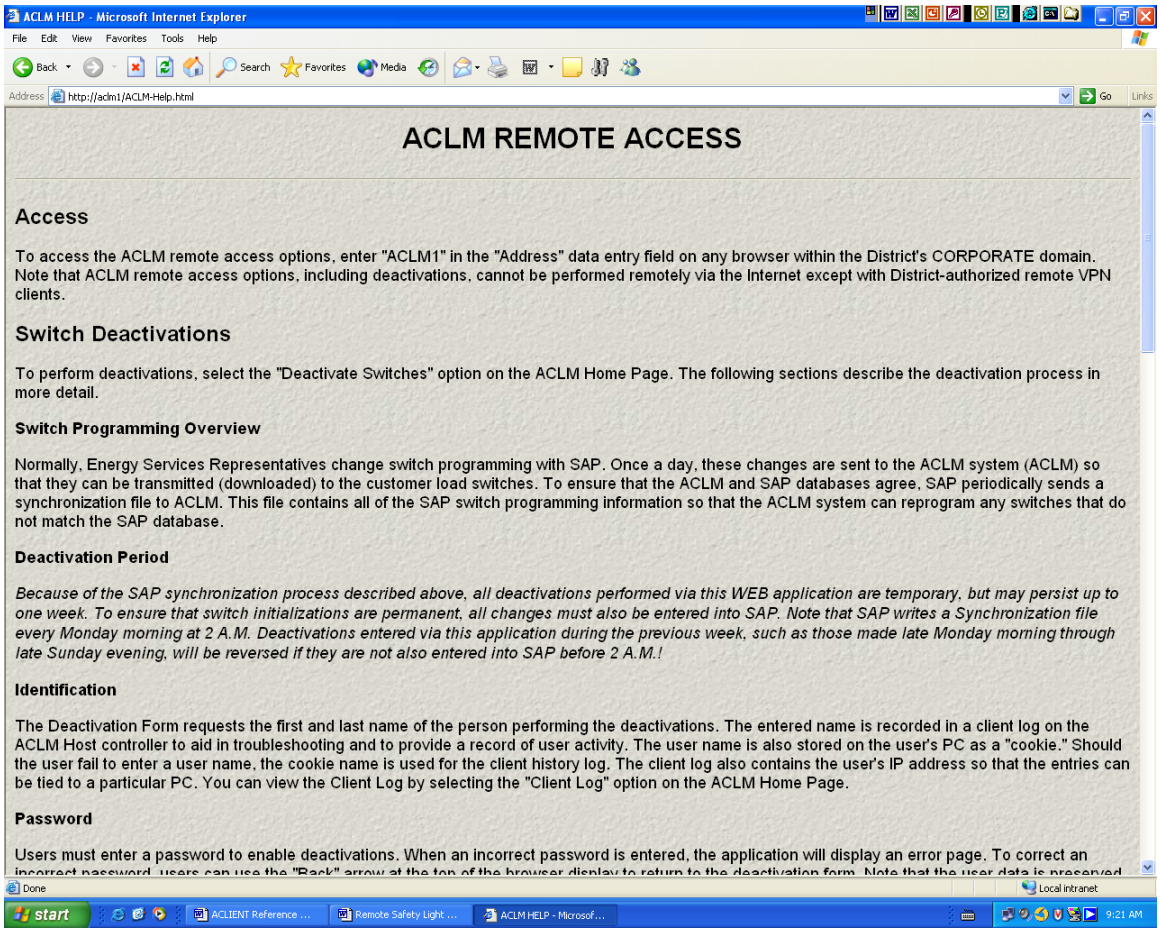

Figure 16. **Help Panel**

Client Log

As shown in Figure 17, the Client Log option on the ACLM Home page displays the log maintained by the ACLIENT application to monitor Client connections and actions. The information displayed in the Client Log is:

1. User Name
2. Name contained in the latest cookie from this Client
3. The IP address of the Client
4. The Data entered by the Client
5. The name of any switch programming files generated by the Deactivate Switches option.

The user name is optional for both the Deactivate Switches and the Safety Lights options but, if it is entered, it is recorded in a cookie on the Client PC so that it can be recovered on a subsequent session (sneaky feature).

The IP address of the Client is also recorded so that the PC used to perform actions can be tracked, if needed. Note that some of the IP addresses are "127.000.000.001." This is the "localhost" IP address, indicating that the user connected from the ACLM HOST computer by entering "localhost" in the address field instead of "ACLM1." This type of entry is usually associated with development or release activity, as further indicated by the suspicious user names and deliberately entered error data.

Note that the data entered by the Client includes all characters, including blanks. The entry of blanks may cause the Client Log to look a little strange. Both the Deactivate Switches and the Safety Lights options ignore embedded blanks in the data entry form. Usually, embedded blanks also indicate application test data rather than typical user data.

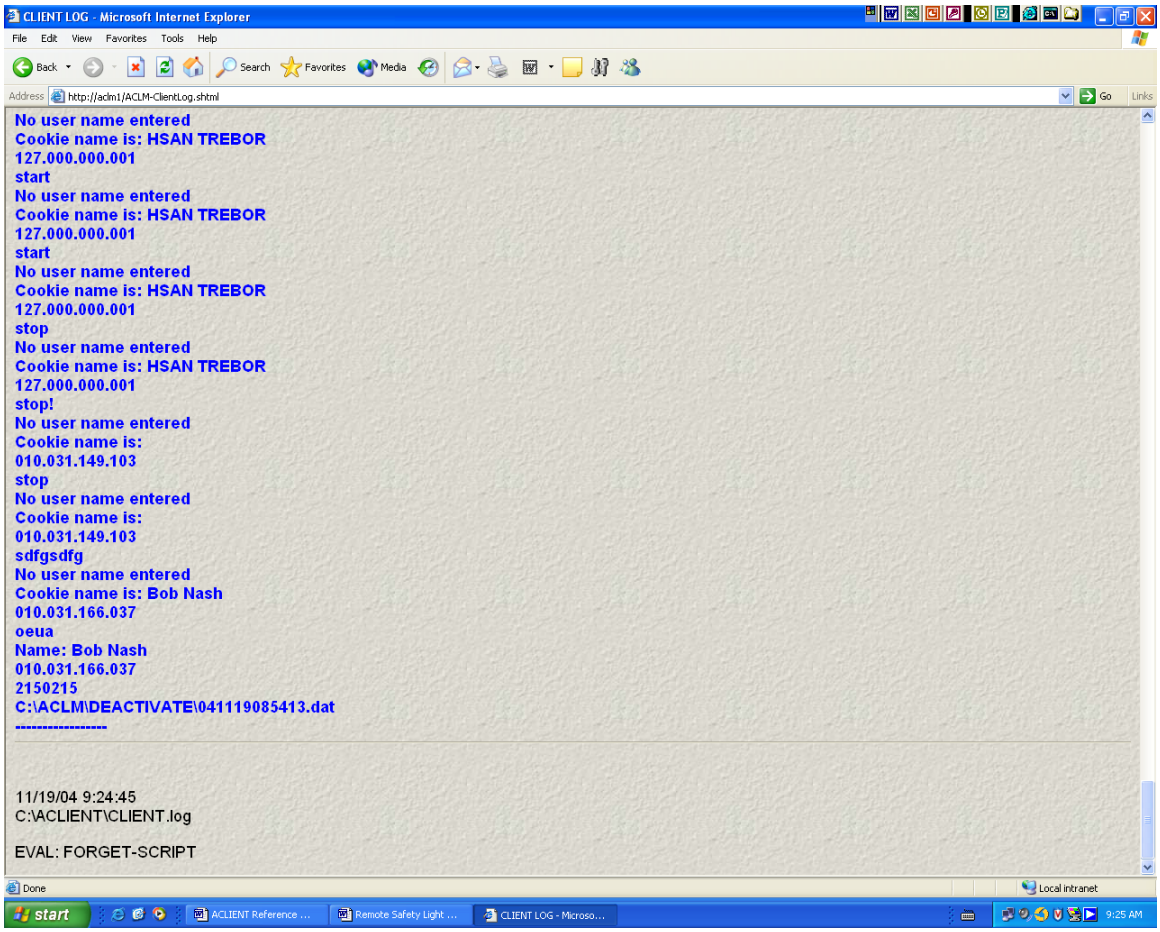

Figure 17. **Client Log**

Information

The Information option on the ACLM Home Page provides information about the operation of the HTTP Server (WEB Server). This Information display is mostly used to verify the operation of the Server and provides a quick way of verifying the Client connection IP and cookie information without having to use password protected options and the Client Log.

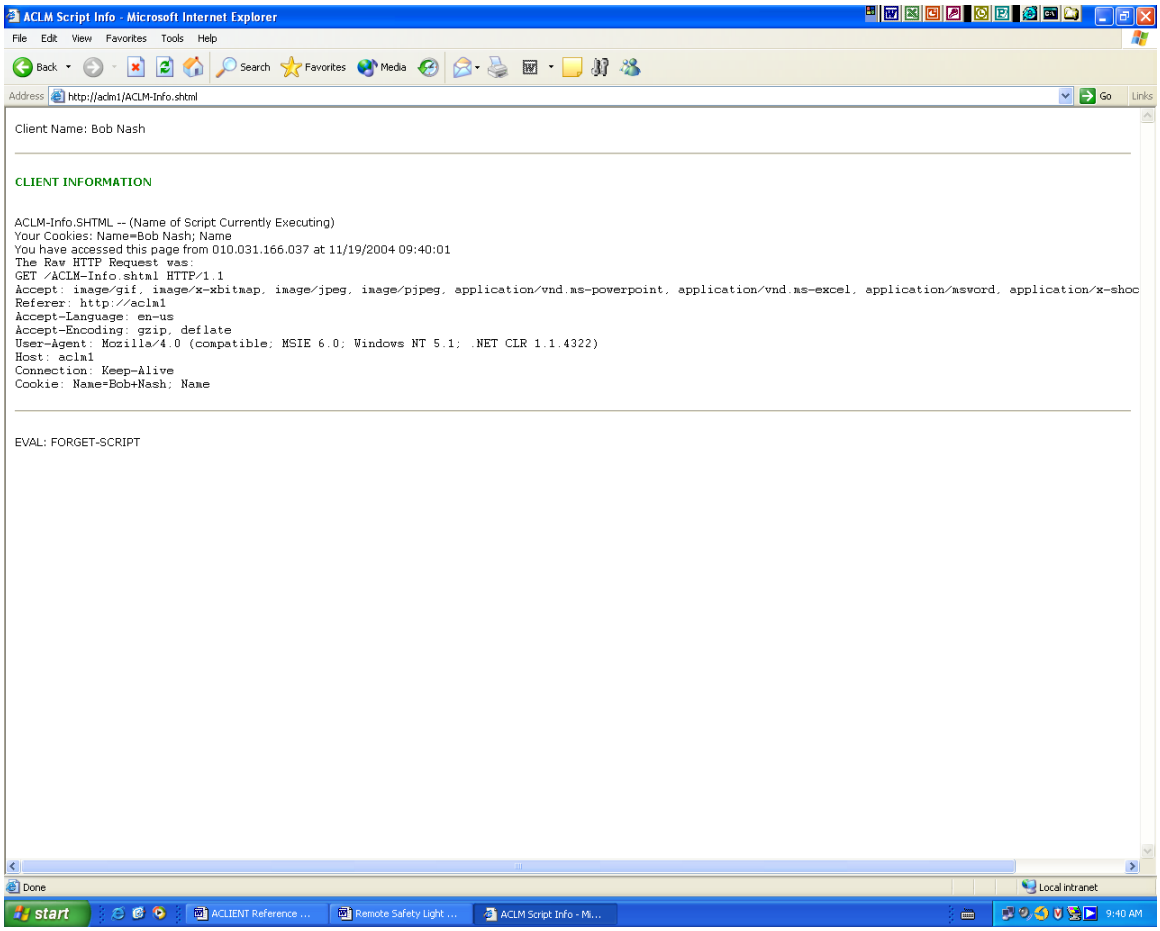

Figure 18. **ACLIENT Information Display**

OPERATION and MAINTENANCE

Overview

The ACLIENT application is normally installed and maintained by the application developer, Bob Nash(x5150), and requires minimal operation and maintenance support. Required support is described below.

Installation

Host

The ACLIENT application resides on the ACLM Host computer, ACLM1 (CORPORATE IP 10.31.149.103). The Host is located in the EMC Telecommunications Room in Rack RRB07. There are no protections against unauthorized access to the ACLIENT application because this is a secure location with access restricted to authorized employees. Also, critical applications are password protected.

Application

The ACLIENT application is installed on the Host in the C:\ACLIENT directory and the WEB subdirectory.

The executable for the HTTP Server and its startup batch file are located in the application's root directory, C:\ACLIENT, and are named WEBSERVER.EXE and STARTSERVER.bat, respectively. These programs and their support files are maintained on the developer's PC and copied to the ACLM1 via the PUBLIC directory. The ACLIENT directory should have no CORPORATE shares, including "read only" access.

The files contained in the ACLIENT and WEB directories are described in more detail in the *Technical Guide* chapter.

Startup

To start the ACLIENT application, execute STARTSERVER.bat in the C:\ACLIENT directory (also see Figure 6).

Do not directly execute WEBSERVER.exe! Directly executing WEBSERVER.exe will start up the WEB server, but will not load the extensions needed for the ACLIENT application.

When the Logix® WEB Server starts up it will display an application window similar to that shown in Figure 6.

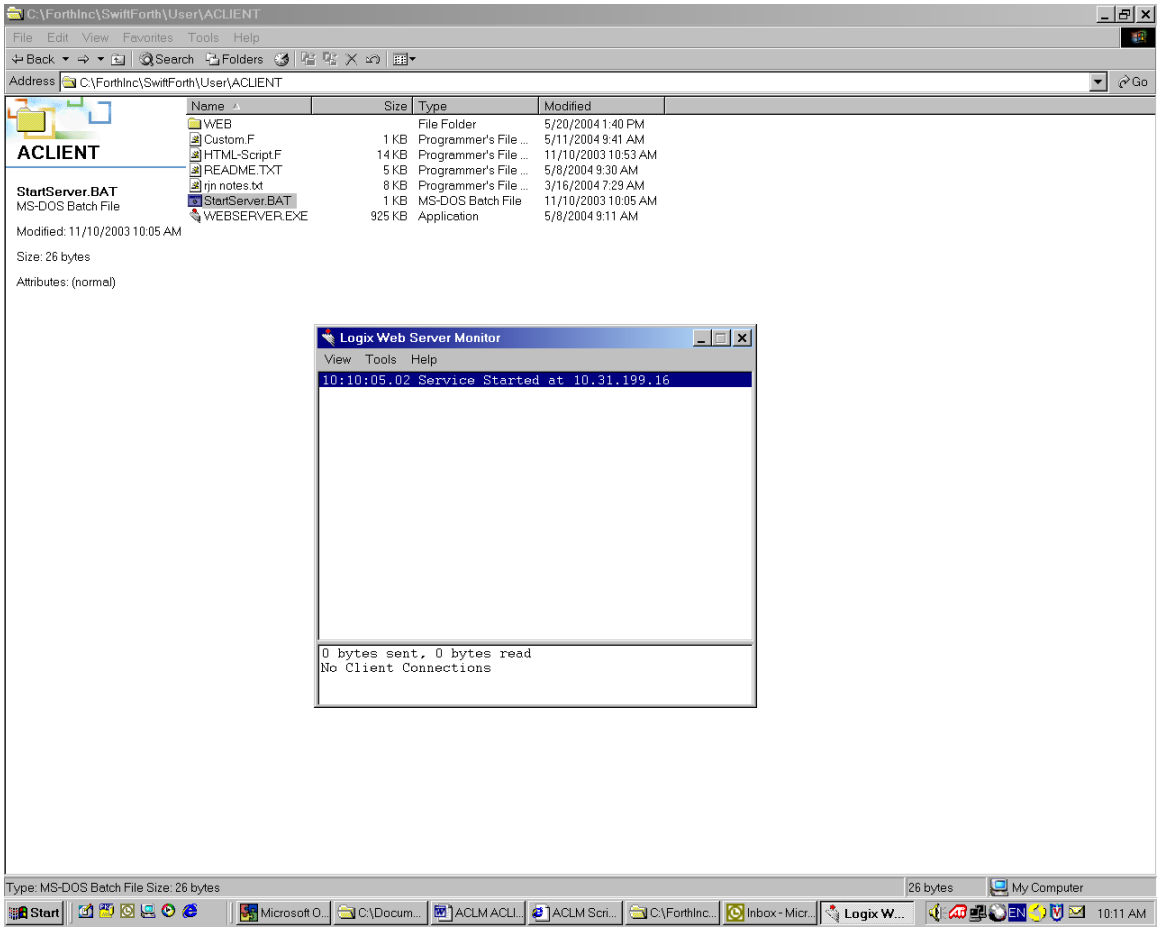

Figure 6. **Logix® WEB Server Monitor Startup**

After the WEB Server Monitor starts up, it can be minimized so that it doesn't distract from the ADA and MONA display screens normally executing on the Host.

Note: The WEB Server Monitor must be started up after any reboots and must be active whenever the ACLM1 Host is operating normally.

Although there are several menus available on the WEB Server Monitor, most of these are of little general interest. Here is a summary of menu options that may be of some use:

1. **Help** - The Help menu provides contact information for Mike Ghan at Logix Controls.
2. **Network Configuration** – The Network Configuration option on the Tools menu shows the TCP Port, The WEB Server IP address and the connection timeout. The TCP Port should be set to 80 (the standard HTTP port). The Web Server IP address should be set to 10.31.149.103, the address of the ACLM1 NIC attached to the CORPORATE network (there are two NICS). The default network connection timeout is 15 minutes. Although this is a little long for normal disabling operations, there is little need to change it.
3. **VIEW** – The View menu should normally show the default view option: "Show All Details." The "Freeze Display" option may also be useful during troubleshooting.

TECHNICAL GUIDE

Overview

Purpose

The purpose of this technical guide is to provide both general and detailed information on the design and operation of the ACLIENT application.

Audience

Some of this information may be useful for those wanting to understand how the application is put together, but not necessarily understand all of the implementation details. This technical information assumes only general familiarity with WEB Servers and CGI programming.

Other information is directed at persons that are familiar with the Swift Forth programming environment but are not familiar with its use as a CGI language with the Logix® WEB Server.

Some information is of interest only to a Developer that is familiar with the general application structure, the operation of the CGI extensions and the Swift Forth programming environment.

The writer apologizes in advance for the mixing of all of these kinds of information. Mercifully, the total amount of information is not large.

Application

ACLIENT is a WEB-based application that uses standard HTML pages and CGI scripts. The CGI scripts use a mix of standard Forth routines (called "words") and custom CGI scripting extensions, also written in Forth, that were developed by Mike Ghan at Logix Controls.

More support information is contained in the *Support* section below. This chapter describes the extensions used by ACLIENT and their automatic compilation at startup. Additional information about the WEB Server is contained in the two help files, as described below.

Appendix A contains the source code printouts for the ACLIENT script, Forth and HTML files.

Structure

ACLIENT Directory

The files in the ACLIENT directory are:

1. **CUSTOM.f** – This file is a Forth source code file containing WEB Server extensions that are compiled at startup. Additional information is contained in this chapter.
2. **HTML-Script.f** – This is a source code file containing WEB Server extensions that have already been incorporated into the WEB Server. It documents Server scripting commands that are available to the developer. It also provides examples for HTML definitions, Forms, Tables, and Forth scripting.
3. **README.txt** – This is the "readme" file, written by Mike Ghan, that is distributed with the Logix® WEB Server.
4. **RJN NOTES.txt** – This file contains additional information about the WEB Server that are not contained in the README.txt file. Most of the information is from correspondence with Mike Ghan (mikeghan@logix-controls.com).
5. **CLIENT.log** – This contains a log of all client activity including the user name, the raw data entered, the date/time of the entry, encapsulated in the name of the deactivation file (e.g., 040523115433.dat for data written on May 5, 2004 at 11:54:33). Note that the ".dat" files are automatically read by the ADA program and transmitted to the Master Station for transmission. After reading these files, ADA deletes them. Thus, these files are transient and the MONA Air Log must be checked to verify that the switch data was actually sent. The ACLM database, accessed via ADA, can also be used to check the programming time of the switch. ADA also maintains an archive file that contains this information. Refer to the *ACLM System Reference Manual* for more information on these facilities.

WEB Subdirectory

The WEB Subdirectory contains all of the HTML, image and CGI script files used to implement the ACLIENT application. The significant files in the WEB subdirectory are:

1. **INDEX.html** – This is the standard HTML home page that is executed whenever a client connects to the WEB Server. It has links to the user options scripts described below.
2. **ACLM-Form.html** – This is the primary ACLIENT user page that provides a form for switch deactivations, as described in the *User Guide* chapter. The processing for this page is described below.
3. **ACLM-SafeForm.html** – This is the primary ACLIENT user page that provides a form for Safety Light activations and deactivations, as described in the *User Guide* chapter. The processing for this page is described below.
4. **ACLM-Help.html** – This user page is the second link provided on the Home page. It provides text information about switch data entry.
5. **ACLM-Info.fs** – This is a script file containing CGI commands and Forth code that is compiled and executed by the client process spawned by the WEB Server. The third option on the Home page links to this script and provides a non-debug version of the information displayed by entering the debug password on the Deactivation Form. Note that this page doesn't display information about the processing of the Deactivation Form, only the general operation of the ACLIENT application.
6. **ACLM-RecvForm.fs** – This is the Forth script file that processes the form information entered via the ACLM-Form.html script. This does most of the ACLIENT "heavy lifting", except for the validation of passwords and switch data. It is described below.
7. **ACLM-SafeRecvForm.fs** – This Forth script file that processes the form information entered via the ACLM-SafeForm.html script. It is cloned from ACLM-RecvForm.fs and does most of the "heavy lifting" for the Safety Light option.
8. **ACLM-OK.fs** – This script is called by ACLM-RecvForm.fs and is the "success" script specified by ACLM-Form.html. It is also described in more detail below.
9. **Common.css** – This is the cascading style sheet specified by most of the ACLIENT HTML and CGI scripts. It specifies defaults for the HTML formatting options. This was retained from the original "demo" application furnished with the WEB Server.

WEB Subdirectory (Cont.)

10. **Riffle1.jpg** – This is the JPEG image used to create the "riffled" gray background on the Home page.
11. **SMUDLogo.gif** – This is GIF image file used to spiff up the Home page.
12. **BAD-URL.fs** – This is the page displayed when the WEB Server when it encounters an error in a user request (e.g., a requested page does not exist). This page should not normally display unless one of the required HTML or script files does not exist (this is unlikely unless there is an installation error)
13. **ThankYou.html** – This is not presently used but is included in the application anyway.

Implementation

Server Extensions

Custom.f - The WEB Server for the ACLIENT application is started up with the StartServer.bat file. This file contains a single line "START WEBServer Custom.f." The "Custom.f" parameter in the startup command line specifies the Forth source code file that WEBServer.exe will load at startup to extend the Server's functionality. Both of these files must reside in the root ACLIENT directory, C:\ACLIENT.

Nothing of much significance is presently performed by Custom.f. : Mostly, it has been retained from the original WEB Server example to illustrate the kinds of things that can be defined and performed at startup.

Presently, Custom.f defines a global variable that all clients can access: it is not used by ACLIENT client processes. Also defined for global usage is a trivial scripting extension to display the WEB Server version.

A routine to generate and format a dated filename is also included in this file for use by all client processes. ***This code and the version code are both used by the ACLIENT application and must be compiled at startup.***

The file also includes "commented out" code to change the TCP port at startup and display a simple message box alert. This shows how some of the Swift Forth Windows calls can be executed at startup.

Client Extensions

In general, most ACLIENT definitions should appear in Custom.f so that client processes do not have to each compile common definitions when they execute. There is one caveat, however. Server extensions cannot include anything that allocates memory that client uses to process transient (user) data. Examples of these would include local VARIABLES, VALUES and user data buffers (e.g., see the SBUF and EBUF buffer allocation in ACLM-OK.fs). Any memory that contains data local to the client process must be allocated and referenced in a client script.

The present ACLIENT application is somewhat deficient in that most definitions are contained in the client scripts. However, the size of the client definitions is relatively small and there is no compelling reason to re-factor the application. For more information on the size allowed for each client process (and how to change it), see the rjn readme.txt file.

Client Processes and Scripts

Processes - A new process is spawned for each client connection. These processes are complete Forth systems with a stacks, buffers, a dictionary and an interpreter. This is what allows Forth definitions to be included in and executed by the CGI scripts.

Scripts - Client CGI scripts are displayed as ".SHTML" (HTML script) in the browser address box but the scripting code is actually contained in files with ".FS" extensions.

The ".FS" extension files contain both CGI commands, as defined in the WEB Server or in Custom.f, or straight Forth definitions and commands. Forth code in these scripts is preceded by a "<%" delimiter and followed by a "%>" delimiter. Although this is a non-standard HTML convention, it seems to be commonly used for this purpose within the Forth community.

Forth Basics

Comments - It may be useful when looking at the application source code to understand what code is active and what code is "commented out." Here is a brief summary of Forth commenting commands:

1. Forth comments can be preceded by a "\" character followed by at least one space before the comment. This form of comment lasts until the end of a line.
2. Text included within parentheses is also ignored (i.e., ANSI standard Forth comments). Note that there must be at least one space between the "(" part of the comment and the comment: the "(" is an executable Forth word that parses to the ")" character.
3. Also supported are "{" and "}" pairs that are not standard, but widely supported in the Forth community. These pairs comment out any code included between them. The "{" is subject to the same requirements as "(", as it is an executable Forth word.

Forth Definitions and Words – It may also be useful to understand the basic way Forth defines and executes commands. Here is a quick guide:

Definitions (Words) - Forth definitions within client scripts are standard: they start with a ":" and end with a ";" . These definitions are compiled by the client processes but are not executed until they are named (called). Routines defined in this way are called "Words" and extend the Forth language: they may be used in the same way as any standard Forth definition and are called by reference.

Conditionals - Words such as [DEFINED], [IF], [ELSE] and [THEN] are conditional operators that are executed interpretively. Conditional Words such as IF, ELSE and THEN are only used within Forth "colon" definitions (these execute to compile conditional branching within a Forth Word).

Comment Display - The ".(" (dot paren) Word and the ")" delimiter also act to display comments during interpretation. The "**."**" (dot quote) Word and the "**"**" (quote) delimiter serve the same purpose within a Forth definition. These work in the same way as the comment Words described above.

ACLM-Form.html

The ACLM-Form HTML script is a standard HTML form that is executed by the client process when the "Deactivation Form" option is selected on the Home page.

The only part of the file that is tricky is the specification of the "success" script with the statement:

<INPUT Type="hidden" Name="success" Value="ACLM-OK.shtml">

This specification is used in the ACLM-RecvForm.fs script as a target script for user input validation. The validation script, ACLM-OK.shtml, is called by the statement:

(Next call the "Success" Page/Script)
REPLY-FORM-SUCCESS? [IF] (True = Success URL Specified?)

The above definition is given in HTML-Script.f.

ACLM-SafeForm.html

The ACLM-SafeForm.HTML script was cloned from the ACLM-Form.HTML script to perform functions appropriate to the Safety Light activation/deactivation option.

ACLM-RecvForm.fs

This script performs the following operations:

1. Writes (appends) data to the client log, CLIENT.log, maintained in the C:\ACLIENT directory (see above).
2. Establishes transient buffers for file names, and for assembling switch deactivation records.
3. Sets client cookie information, including user name and expiration dates
4. Calls ACLM-OK.fs to validate user data entered in ACLM-Form.html.
5. Writes the switch deactivation file into the C:\ACLM\DEACTIVATION directory (the file is automatically managed by the ADA program afterward). This file uses the ACLM "standard" format (i.e., crlf delimited switch programming parameters). The file names use the generation date and time, as described in the source code.

ACLM-SafeRecvForm.fs

This script was cloned from ACLM-RecvForm.fs and performs functions similar to those given above, but for the Safety Lights option. Here is a summary of the functions performed by this script:

Writes (appends) data to the client log, CLIENT.log, maintained in the C:\ACLIENT directory (see above).
Establishes a transient buffer for activation commands (i.e., start and stop).
Sets client cookie information, including user name and expiration dates
Calls ACLM-SafeOK.fs to validate user data entered in ACLM-SafeForm.html.
Writes the Safety Light activation and deactivation flag files, SAFE.flg and STOP.flg, to the C:\ACLM\SAFE directory (these file are automatically managed by the ADA program afterward.

ACLM-OK.fs

This script performs the following:

1. Validates user passwords
2. Establishes transient buffers for processing switch data entered from ACLM-Form.html and for assembling error strings.
3. Validates user data
4. Generates an HTML "results" page

ACLM-SafeOK.fs

This script performs functions similar to that given above for ACLM-OK.fs, but as they apply to the Safety Lights option.

Support

Although it has been in use since the mid 70's (about the same time as Unix), Forth is still a little known language that, if acknowledged at all, is not considered a "major" or "mainstream" language like C, C++, Java or Basic.

Notwithstanding the lack of widespread recognition as a mainstream language, Forth is an ANSI Standard language that is widely supported, both by Forth, Incorporated (WWW.Forth.com), and a number of independent contractors and user groups (WWW.Forth.org). Forth, Incorporated, has been in continuous operation since 1979 and a District "partner" vendor since the early 1990's.

The Logix® WEB Server is written in Swift Forth by Mike Ghan, as are most of the Server extensions. Although Mike cannot release the proprietary source code for the executable, he does provide source code listings for many of the CGI scripting extensions that he has incorporated into the Server. Many of these scripting extensions are based on extensions first proposed by Bernrd Paysan and widely used for CGI extensions to various Forth implementations.

Developers can extend the Server functionality by entering the name of an extension file, consisting of Forth source code, as a command line parameter at startup.

Although Mike Ghan does not officially support the WEB Server, he has fixed several minor bugs within several hours and is generally available, as a courtesy, to answer questions.

Questions can be posted to him directly via the address given in the previous chapter, but are best posted via the SF Talk user group so that all of the Swift Forth user community can benefit from his responses.

The WEB Server has been in continuous use for several years, supporting the Logix Controls WEB site and several Swift Forth user applications. In short, it seems to be very robust and bug-free and it is supported via a number of mechanisms.

APPENDIX A

SOURCE CODE LISTINGS

Custom.f

```
{ ============================================================
  Custom Forth Routines for WebServer

  Changes:
  Created 12/10/2002 by Mike Ghan
  Modified 05/11/2004 by Bob Nash

============================================================ }

VARIABLE TRANSACTION#  \ Example of a Global Variable

HTML DEFINITIONS

: SHOW-VERSION  ( -- )
  BREAK RED BOLD <ATTR> .VERSION   </ATTR> BREAK ;

: DATEDFNAME ( -- a n )  TIME&DATE  2000 - 0 5 0 DO 100 1 M*/ ROT M+ LOOP
  <# # # # # # # # # # # # #> ;

/FORTH

\ 81 WEB-MASTER TCP-PORT ! \ Set HTTP Port

FALSE [IF] \ Test Message Box - Be Careful, No Window Exists Yet
S" Port altered to " PAD ZPLACE
WEB-MASTER TCP-PORT @ (.) PAD ZAPPEND
0 ( No HWND ) PAD  Z" Testing"
MB_SYSTEMMODAL MB_OK OR  MessageBox DROP
[THEN] \ End Test
```

README.txt

WebServer.EXE written by Mike Ghan, Logix Industrial Controls

This WebServer is written entirely in SwiftForth. The WebServer will serve up simple HTML pages as well as Forth script files. The zip file contains the WebServer with a few samples. Create a suitable directory and unzip (be sure to retain the relative directory stucture in the zip file).

All web files must reside in the WEB subdirectory tree of the server executable (assuming WebServer.EXE resides in C:\SERVER\, all web pages would be in C:\SERVER\WEB\). I do test for malformed URLs and buffer overflows so a user *should* not be able to access other directories or crash this server. *Please* let me know otherwise.

Forth script files have the .FS extension (ex. Primes.FS) but are accessed as .SHTML (Script HTML) files from a client browser (ex. http://www.yada.com/Primes.shtml). When a script file is served up, the Forth script file is interpreted (just as in INCLUDE). It is possible to define Forth definitions and execute them. See Primes.FS for an example.

The WebServer.EXE will allow command line parameters just as SwiftForth does. This will allow you to extend the server by including Forth source (eg typing WebServer CUSTOM.F from the command line would INCLUDE the CUSTOM.F example file and start the server). This is resident code as opposed to any Forth compiled during a script which is transient. See the example batch file StartServer.Bat

Launch the WebServer and point your browser to http://localhost for a few examples.

NOTE! The Forth script engine provides no safeguards to prevent disk access, malicious code etc. It obviously is suitable for deployment in tightly controlled environments only.

The WebServer also has support for the POST method and Cookies. My examples are a bit rough (not realistic and contain much test code): point your browser at FormTest.html, fill in the fields and submit the form. RecvForm.FS will process the POSTed form and write a file named COMMENT.TXT with the contents of the form's Comment field. The First and Last names are concatenated and saved as a cookie named Name. Likewise, the email field is saved as a cookie named Email. Since our test HTML form contains a "success=RecvFrom.SHTML" page reference (more typically a simple ThankYou.HTML page). the script RecvFrom.FS is then served as a response to the browser.

Subsequent browser access to RecvFrom.SHTML will display the cookies.

README.txt (Cont.)

Here are the existing supported content types. All except shtml are simply sent "as is" to the client browser.

```
S" text/html"                MIME: html
S" text/html"                MIME: htm
S" image/gif"                MIME: gif
S" image/jpeg"               MIME: jpg
S" image/png"                MIME: png
S" application/x-javascript"  MIME: js    \ javascript
S" application/pdf"          MIME: pdf
S" application/zip"          MIME: zip
S" text/plain"               MIME: txt   \ our default
```

You can add more (place in a .F file included from the command line when the server is launched):

```
---------------------------------------
MIMES DEFINITIONS

\ Mime type from browser      URL extension
S" application/foobar"        MIME: foo

/FORTH FORTH DEFINITIONS

---------------------------------------
```

README.txt (Cont.)

```
Some additional "documentation":

\ Get the entire content received from the Client form.
GET-CLIENT-CONTENT  ( -- addr count )

\ Get the Content Value from Name=Value Pair received from the Client form.
GET-CLIENT-CONTENT-VALUE  ( NameAddr count -- ValueAddr count )


\ Cookie Template:  NAME=VALUE; expires=DATE; path=PATH; domain=DOMAIN_NAME;
secure
\ Example: MeLove=Cookie%20Monster; expires=Thursday, 01-Jan-98 12:00:00 GMT"
\ Define a Cookie to be Sent to the Client, Usually Name=Value pair.
SET-CLIENT-COOKIE-NAME  ( addr count -- )

\ Append a Value to the Client's Cookie
SET-CLIENT-COOKIE-VALUE  ( addr count -- )

\ If Expiration is not set, the cookie is valid during the current session only.
SET-CLIENT-COOKIE-EXPIRE  ( secondsOffset daysOffset -- )

\ Get All the Cookie Name=Value Pair(s) received from the Client.
GET-CLIENT-RAW-COOKIE  ( -- addr count )

\ Get the Cookie Value from Name=Value Pair received from the Client.
\ ex    S" PhoneNumber" GET-CLIENT-COOKIE-VALUE
GET-CLIENT-COOKIE-VALUE  ( NameAddr count -- ValueAddr count )


GET-HTML-ARGS  ( -- Addr Cnt )
GET-HTML-URL  ( -- Addr Cnt )
SET-REFRESH  ( #secs -- )
GET-CLIENT-IP  ( -- Addr Cnt )



Best Regards,
Mike Ghan
```

RJN NOTES.txt

RJN Notes about Mike Ghan's Web Server 10Mar04
==========

Usage
---
1. Execute WEBSERVER.exe, HTML and script (*.FS) files are contained in the WEB subdirectory.
2. Users accessing the server (e.g., via HTTP://10.31.199.16) will see INDEX.html contained in the WEB subdirectory.

Cookies
---
1. The cookie info is stored in a file named something like: C:\Documents and Settings\bnash\cookies\bnash@10.31.199.16[1].txt
2. The form text is stored in the web server root directory as COMMENT.txt

Misc. Info (best of correspondence below)
---
1. Q: Because your server is a turnkeyed app, approximately how much dictionary is available for extensions at startup? I assume there is also a limit on the Forth definitions contained in the client CGI scripts. Can you tell me approximately how much that is?
   A: At startup, all the memory is available just as in a SwiftForth console session. A client script is limited, the default is 50K, but because it is a VALUE, you can alter it at startup: 75000 TO /SCRIPT-DICT
2. Q: What is the latest version of your HTTP Server?
   A: Latest version is 11/11/2003 which fixed a few problems when INCLUDEing forth source when the server was launched. You can dowload it from: **http://www.logix-controls.com/SwiftForth\Webserver\WebServer.zip**
3. Q: Can you provide a simple explanation of how the HTTP requests from a client are formatted in the socket data?
   A: HTTP requests are fairly straight forward - keep reading until a pair of Cr/Lf are received, the connection times out, or until your buffer is full (important). I've implemented the server in SWOOP with each client allocating buffer space on the fly. I gleaned a few ideas from **Bernd Paysan's web server: http://www.jwdt.com/~paysan/httpd-en.html**

RCLIENT.log

```
Name: Robert Nash
127.000.000.001
21502152012374
C:\ACLM\DEACTIVATE\040524060550.dat
-----------------
Name: Roger Rabbit
010.031.199.016
2149450
2118091
C:\ACLM\DEACTIVATE\040524060847.dat
-----------------
```

Index.html

```
<HTML><HEAD>
<TITLE>ACLM Deactivation</TITLE>
</HEAD>
<BODY background="riffle1.jpg">

<P><IMG BORDER="0" SRC="SMUD Logo.gif" WIDTH="320" HEIGHT="64" ALT="by Robert
Nash" HSPACE=5></P>
<H2>ACLM Emergency Deactivations</H2>
<HR>

<TABLE Border="0" STYLE="font-size: 135%">
  <TR>
    <TD><A HREF="ACLM-Form.html" >Deactivation Form</A></TD>
  </TR>
  <TR>
    <TD><A HREF="ACLM-HELP.html" >HELP</A></TD>
  </TR>
  <TR>
    <TD><A HREF="ACLM-Info.shtml" >Information</A></TD>
  </TR>
</TABLE>


</BODY>
</HTML>
```

ACLM-Info.html

```
<HTML>
<HEAD>
<TITLE>ACLM Script Info</TITLE>
<META http-equiv="Pragma" content="no-cache">
<LINK rel="stylesheet" type="text/css" href="Common.css">
</HEAD>
<BODY>

<P>

<% ( ############### Start Forth ############### )

{ -------------------------[ Client Definitions  ]-------------------------

NOTES:
1. This is the non-secret version of ACLM-OK
 ------------------------------------------------------------------------- }

\ ----- clunky way to handle pesky HTML quotes
CREATE PRE-DATA CHAR * STRING <PRE TITLE="Raw HTTP Request">*

: .ME  ( -- )   S" ACLM-Info.SHTML -- " TYPE ;

\ ----- clunky way to handle pesky HTML quotes
CREATE PRE-DATA CHAR * STRING <PRE TITLE="Raw HTTP Request">*
```

ACLM-Info.html (Cont.)

```
: .CLIENT-INFO  ( -- )
   S" <HR>" TYPE
   BREAK S" <H4>" TYPE  S" CLIENT INFORMATION " TYPE  S" </H4>" TYPE
   BREAK BREAK .ME  S" (Name of Script Currently Executing)" TYPE
   BREAK S" Your Cookies: " TYPE  GET-CLIENT-RAW-COOKIE DECODE-HTTP TYPE
   BREAK S" You have accessed this page from " TYPE
      GET-CLIENT-IP TYPE S"  at " TYPE .TS
   BREAK PRE-DATA COUNT TYPE \ Add Tooltip!
   S" The Raw HTTP Request was:" TYPE
      CR SCRIPT-CLIENT USING WEB-CLIENT-RES READ-BUFR LCOUNT TYPE
   S" <HR>" TYPE   S" </PRE>" TYPE ;

\ ------------------------------------------------------------------------

\ Note: This processes the cookie password, not the entered password!
[DEFINED] GET-HTML-ARGS
   [IF] ( Logix Server Extensions)
      S" Name" GET-CLIENT-COOKIE-VALUE  DUP
      [IF]
         CR S" Client Name: " TYPE  TYPE
      [ELSE]
         2DROP ( bad name $)  CR S"  Anonymous User" TYPE
      [THEN]  .CLIENT-INFO
   [ELSE] ( No Extensions -- unlikely, but ...)
      .( Server Extensions Not Defined)
   [THEN]

(              ################################              End              Forth
################################ ) %>

</P>
</BODY>
</HTML>
```

ACLM-Form.html

```
<HTML>
<HEAD>
<TITLE>ACLM Input Form</TITLE>
</HEAD>

<BODY>
<H3>ACLM Switch Deactivation Form</H3>
<HR>
<STRONG>Please enter your deactivation information</STRONG>
   <FORM Method="POST" Action="ACLM-RecvForm.SHTML">
     <! ACLM-OK.FS executes after ACLM-RecvForm.FS>
     <INPUT Type="hidden" Name="success" Value="ACLM-OK.shtml">

       <TABLE Border="0" Width="536">
       <TR>
         <TD Width="65"><FONT Size="2" Face="Arial">Name</FONT></TD>
         <TD Width="51"><FONT Size="2" Face="Arial"><INPUT Type="text" Size="14"
Maxlength="40" Name="first"></FONT></TD>
         <TD Width="156"><DIV Align="left">
         <P><FONT Size="2" Face="Arial">First</FONT></TD>
         <TD Width="211"><FONT Size="2" Face="Arial"><INPUT Type="text" Size="25"
Maxlength="50" Name="last"></FONT></TD>
         <TD Align="right" Width="33"><FONT Size="2" Face="Arial">Last</FONT></TD>
       </TR>
       <TR>
         <TD Width="65"><FONT Size="2" Face="Arial">Password</FONT></TD>
         <TD Colspan="4" Width="463"><FONT Size="2" Face="Arial"><INPUT
Type="password" Size="20" Maxlength="40" Name="password"></FONT></TD>
       </TR>
       <TR>
         <TD Valign="top" Width="65"><FONT Size="2" Face="Arial">Switches</FONT></TD>
         <TD Colspan="4" Width="463"><FONT Size="2" Face="Arial"><TEXTAREA
Name="switches" Rows="5" Cols="7" Wrap="physical"></TEXTAREA> </FONT></TD>
       </TR>
       <TR>
         <TD Width="65"></TD>
         <TD Colspan="2" Width="211"><FONT Size="2" Face="Arial"><INPUT Type="submit"
Value="Submit"></FONT></TD>
         <TD Align="right" Colspan="2" Width="248"><FONT Size="2" Face="Arial"><INPUT
Type="reset" Value="Reset Form"></FONT></TD>
       </TR>
     </TABLE>

   </FORM>
</BODY>
```

ACLM-Form.html (Cont.)

```
<ADDRESS>
Bob Nash <A HREF="mailto:bnash@SMUD.org">bnash@SMUD.org</A>
</ADDRESS>
</HTML>
```

ACLM-SafeForm.html

```
<HTML>
<HEAD>
<BODY background="riffle1.jpg">
<TITLE>SAFETY LIGHT FORM</TITLE>
</HEAD>
<FONT Size="4" Face="Arial">
<B><H1><div align="center">SAFETY  LIGHTS</div></H1></B><HR>

<STRONG>Please enter your safety user information</STRONG>
   <FORM Method="POST" Action="ACLM-SafeRecvForm.SHTML">
     <! ACLM-OK.FS executes after ACLM-SafeRecvForm.FS>
     <INPUT Type="hidden" Name="success" Value="ACLM-SafeOK.shtml">

       <TABLE Border="0" Width="536">
       <TR>
         <TD Width="65"><FONT Size="2" Face="Arial">Name</FONT></TD>
         <TD Width="51"><FONT Size="2" Face="Arial"><INPUT Type="text" Size="14"
Maxlength="40" Name="first"></FONT></TD>
        <TD Width="156"><DIV Align="left">
        <P><FONT Size="2" Face="Arial">First</FONT></TD>
        <TD Width="211"><FONT Size="2" Face="Arial"><INPUT Type="text" Size="25"
Maxlength="50" Name="last"></FONT></TD>
        <TD Align="right" Width="33"><FONT Size="2" Face="Arial">Last</FONT></TD>
       </TR>
       <TR>
         <TD Width="65"><FONT Size="2"
```

ACLM-SafeForm.html (Cont.)

```
Face="Arial">Password</FONT></TD>
        <TD Colspan="4" Width="463"><FONT Size="2" Face="Arial"><INPUT
Type="password" Size="20" Maxlength="40" Name="password"></FONT></TD>
      </TR>
      <TR>
        <TD Valign="top" Width="65"><FONT Size="2" Face="Arial">Actions</FONT></TD>
        <TD Colspan="4" Width="32"><FONT Size="2" Face="Arial"><TEXTAREA
Name="Actions" Rows="1" Cols="8" Wrap="physical"></TEXTAREA> </FONT></TD>
      </TR>
      <TR>
        <TD Width="65"></TD>
        <TD Colspan="2" Width="211"><FONT Size="2" Face="Arial"><INPUT Type="submit"
Value="Submit"></FONT></TD>
        <TD Align="right" Colspan="2" Width="248"><FONT Size="2" Face="Arial"><INPUT
Type="reset" Value="Reset Form"></FONT></TD>
      </TR>
    </TABLE>

  </FORM>
</BODY>
</HTML>
```

ACLM-Help.html

```
<HTML>
<HEAD>
<TITLE>ACLM Help</TITLE>
</HEAD>
<P><FONT Size="4" Face="Arial">

<H2>Deactivation</H2>

The deactivations performed by ACLIENT are temporary.  Normally, changes to customer switch
programming are performed by an SAP application.
Once a day, the SAP application sends an update file to the ACLM System. The update file
contains information about the customer option changes performed during the day; the ACLM
System uses this SAP extract information to program the customer load switches.
The only "official" changes to customer switch programming are via the SAP application and the
daily extract file.  To ensure that the database maintained by the ACLM System agrees with the
SAP database, SAP periodically generates a synchronization file containing all of the switches in
the SAP database.  The ACLM System used this file to reprogram any switches in its database
that do not agree with the switch information in the SAP synchronization file.
Thus, any temporary switch disabling (initialization) performed with ACLIENT will be reversed
when the SAP and the ACLM System synchronize databases: to make ACLIENT switch
initializations permanent, the changes must be entered into SAP.

<H2>Access</H2>

To access the ACLIENT application, enter "HTTP://ACLM1" in the "Address" data entry field on
any browser within the District's CORPORATE domain.  Note that the ACLIENT application cannot
be accessed via the Internet except via District-authorized remote clients.  Normally, such remote
deactivations are both unnecessary and undesirable.
To perform deactivations, select the "Deactivation Form" option.
```

ACLM-Help.html (Cont.)

<H2>Name Entry</H2>

Users are requested to enter their first and last names. These are recorded in a client log on the ACLM Host controller to aid in troubleshooting and to provide a record of user activity.
The user name is also stored on the user's PC as a "cookie." Should the user fail to enter a user name, the user name from the user's cookie is entered in the client history log. The client log also contains the user's IP address so that entries can be tied to particular PCs.

<H2>Password</H2>

Users must enter a password to enable deactivations. If an incorrect password is entered, ACLIENT will display an error page. To correct an incorrect password, users can use the "Back" arrow at the top of the browser display to return to the deactivation form. Note that the user data is preserved so that the data does not have to be re-entered after a password error.

<H2>Switch Data Entry</H2>

The Deactivation Form provides a small text entry field for entry of switch (cycler) numbers. The data entry form is seven characters wide to help users visually validate the size of switch numbers (note: all switch numbers are seven digits long).
The data entry field does not provide any protection against entry of non-numeric or incorrect switch numbers: switch data is validated after it is submitted.
Switch data validation is as follows:

1. Switch numbers may be entered in a continuous sequence or they may be entered with one or more carriage returns between them, 2. Non-numeric data between switch entries is ignored (e.g., carriage returns, blanks, commas and periods), 3. Short switch numbers will be ignored, but will be shown as errors
4. Non-numeric characters at the start and end of the switch entry field are ignored, 5. Switches outside the District's switch range of 20000000 to 2400000 are ignored but are shown as errors

ACLM-Help.html (Cont.)

```
<H2>Validation</H2>

After the Deactivation Form is complete, users can submit their deactivation candidates by
clicking on the "Submit" button.
Switch entries are then validated and the results are displayed.
Note that the validation page shows both the valid deactivation entries and the errors found
during processing.
Valid switches are displayed in blue and errors are displayed in red.

</FONT></P>
</BODY>
</HTML>
```

ACLM-RecvForm.fs

```
HTML ( Access HTML Vocabulary )

\ ----------------------------[ Client Log File ]----------------------------

: RAW>CLOG  ( a n -- )   C" CLIENT.log" ~>>FILE TYPE CONSOLE ;  \ no crlf
: >CLOG  ( a n .. )   <CRLF> COUNT RAW>CLOG  RAW>CLOG ;


\ ----------------------------[ Switch Data File ]----------------------------

\ ----- buffer for data file name
64 CONSTANT #FBUF
CREATE FBUF #FBUF ALLOT

\ ----- create dated filename for switch data file
\ (e.g., 040514083417.dat for May 14th, 2004 at 08:34:17)

S" C:\ACLM\DEACTIVATE\" FBUF PLACE  DatedFName FBUF APPEND  S" .dat" FBUF APPEND

: $>DFILE  ( a n -- )   FBUF ~>>FILE TYPE CONSOLE ;  \ add $ to deactivate file

4096 CONSTANT #DBUF   \ data buffer for deactivate images
CREATE DBUF #DBUF /ALLOT

: !DBUF  ( a cnt -- )   \ store a $ in the switch data buffer
   DBUF DUP C@ IF APPEND ELSE PLACE THEN
   S"  01 000 000 01" DBUF APPEND  <CRLF> COUNT DBUF APPEND ;

\ ----------------------------[ Client Cookie ]----------------------------

\ ----- set name value in cookie
S" Name" SET-CLIENT-COOKIE-NAME
S" First" GET-CLIENT-CONTENT-VALUE >SPAD SPACE>SPAD
S" Last"  GET-CLIENT-CONTENT-VALUE SPAD+
SPAD COUNT DUP 1 > [IF] SET-CLIENT-COOKIE-VALUE [ELSE] 2DROP [THEN]

\ S" password" SET-CLIENT-COOKIE-NAME
\ S" password" GET-CLIENT-CONTENT-VALUE SET-CLIENT-COOKIE-VALUE

0 ( time offset in seconds )  10 ( days ) SET-CLIENT-COOKIE-EXPIRE
```

ACLM-RecvForm.fs (Cont.)

```
\ ----------------------[ Write user data to a log file ]----------------------

\ ----- write Client name to log file
SPAD COUNT DUP 1 > NOT [IF]
   2DROP S" No user name entered"  >CLOG
   S" Cookie name is: " >CLOG
   S" Name" GET-CLIENT-COOKIE-VALUE RAW>CLOG
[ELSE]
   S" Name: " >CLOG  RAW>CLOG
[THEN]

GET-CLIENT-IP DUP [IF] >CLOG [ELSE] 2DROP [THEN]

\ ----- write Client raw switch data to log file
S" switches" GET-CLIENT-CONTENT-VALUE DUP 0= [IF]
   2DROP S" No data entered"
[THEN] >CLOG

\ ----- echo data filename to capture date/time
FBUF COUNT >CLOG   S" ----------------" >CLOG

( Next Call the "Success" Page/Script -- see HTML-Script.f )
REPLY-FORM-SUCCESS? [IF] ( True = Success URL Specified? )
   ( Note: You can't use BREAK here!)
   DBUF COUNT DUP [IF]
      $>DFILE
      CR .( Switch data written to: )  FBUF COUNT TYPE
   [ELSE]
      2DROP  CR .( No switch data to write!)
   [THEN]
\   CR CR .( ACLM-RecvForm.SHTML -- Form Processing is Complete)
[ELSE] ( else )
\   PLAY-WARNING \ Play Windows warning sound on Server
   <HTML>
   <HEAD>
   <TITLE>Missing Success Page</TITLE>
   <META http-equiv="Pragma" content="no-cache">
   </HEAD>
   <BODY>
   <P>
   <HR>
   <STRONG>Missing Success Page! (Data Dump Below)</STRONG>
```

ACLM-RecvForm.fs (Cont.)

```
<% ( ############################ Start Forth
################################## )

   BREAK .( URL: )             GET-HTML-URL TYPE
   BREAK .( HTTP Arguments: )   GET-HTML-ARGS TYPE
   BREAK
   .( <PRE TITLE="Raw HTTP Request">)  \ Add Tooltip!
   .( The Raw HTTP Request was:)
   CR SCRIPT-CLIENT USING WEB-CLIENT-RES GET-HEADERS TYPE
   CR .( The Raw HTTP Content was:)
   SCRIPT-CLIENT USING WEB-CLIENT-RES GET-CONTENT TYPE
   .( </PRE>)

( ############################### End Forth
################################ ) %>

   <HR>
   </P>
   </BODY>
   </HTML>
[THEN]
```

ACLM-SafeRecvForm.fs

```
HTML ( Access HTML Vocabulary )


\ ----------------------------[ Client Log File ]---------------------------

: RAW>CLOG  ( a n -- )   C" CLIENT.log" ~>>FILE TYPE CONSOLE ;  \ no crlf
: >CLOG  ( a n .. )   <CRLF> COUNT RAW>CLOG  RAW>CLOG ;


\ ------------------------------[ Client Cookie ]-----------------------------

\ ----- set name value in cookie
S" Name" SET-CLIENT-COOKIE-NAME
S" First" GET-CLIENT-CONTENT-VALUE >SPAD SPACE>SPAD
S" Last"  GET-CLIENT-CONTENT-VALUE SPAD+
SPAD COUNT DUP 1 > [IF] SET-CLIENT-COOKIE-VALUE [ELSE] 2DROP [THEN]

\ S" password" SET-CLIENT-COOKIE-NAME
\ S" password" GET-CLIENT-CONTENT-VALUE SET-CLIENT-COOKIE-VALUE

0 ( time offset in seconds )  10 ( days ) SET-CLIENT-COOKIE-EXPIRE


\ ---------------------[ Write user data to a log file ]----------------------

\ ----- write Client name to log
SPAD COUNT DUP 1 > NOT [IF]
   2DROP S" No user name entered"  >CLOG
   S" Cookie name is: " >CLOG
   S" Name" GET-CLIENT-COOKIE-VALUE RAW>CLOG
[ELSE]
   S" Name: " >CLOG  RAW>CLOG
[THEN]

\ ----- write Client IP to log
GET-CLIENT-IP DUP [IF] >CLOG [ELSE] 2DROP [THEN]

\ ----- write Client raw action data to log
S" actions" GET-CLIENT-CONTENT-VALUE DUP 0= [IF]
   2DROP S" No action data entered"
[THEN] >CLOG
```

ACLM-SafeRecvForm.fs (Cont.)

```
\ ----- echo data filename to capture date/time
\ AFBUF COUNT >CLOG   S" ----------------" >CLOG

( Next Call the "Success" Page/Script -- see HTML-Script.f )
REPLY-FORM-SUCCESS? [IF] ( True = Success URL Specified? )
   CR CR .( ACLM-SafeRecvForm.SHTML -- Form Processing is Complete)
[ELSE] ( else )

\   PLAY-WARNING \ Play Windows warning sound on Server
   <HTML>
   <HEAD>
   <TITLE>Missing Success Page</TITLE>
   <META http-equiv="Pragma" content="no-cache">
   </HEAD>
   <BODY>
   <P>
   <HR>
   <STRONG>Missing Success Page! (Data Dump Below)</STRONG>

<% ( ############################ Start Forth
################################## )

   BREAK .( URL: )             GET-HTML-URL TYPE
   BREAK .( HTTP Arguments: )   GET-HTML-ARGS TYPE
   BREAK
   .( <PRE TITLE="Raw HTTP Request">)  \ Add Tooltip!
   .( The Raw HTTP Request was:)
   CR SCRIPT-CLIENT USING WEB-CLIENT-RES GET-HEADERS TYPE
   CR .( The Raw HTTP Content was:)
   SCRIPT-CLIENT USING WEB-CLIENT-RES GET-CONTENT TYPE
   .( </PRE>)

( ############################## End Forth
################################ ) %>

   <HR>
   </P>
   </BODY>
   </HTML>
[THEN]
```

ACLM-OK.fs

```
<HTML>
<HEAD>
<TITLE>ACLM Script Testbed</TITLE>
<META http-equiv="Pragma" content="no-cache">
<LINK rel="stylesheet" type="text/css" href="Common.css">
</HEAD>
<BODY>

<P>

<% ( ################ Start Forth ################ )

\ NOTE! This script is called by (and executes after) ACLM-RecvForm.FS

\ ----------------------------[ Client ID  ]-----------------------------

: .AOK  ( -- )   S" ACLM-OK.SHTML -- " TYPE ;
BREAK .AOK .( Executing ...)


{ -------------------[ Switch Buffer, Navigation, Display ]-----------------

SBUF contains the data entered by the client browser
START returns an address within SBUF that represents the start of a switch
   candidate to be processed.  This address is incremented as switches
   are processed or bad data is encountered
+START bumps the START address to move to the next switch candidate
NEXT-CHAR advances the START address by one character, primarily to skip
   leading non-numeric characters
#END returns the address of the cell just past the end of SBUF data
#LEFT returns the number of characters left in SBUF
MORE? returns a true flag if there are more characters left in SBUF, based
   on the current value of START
SWITCH$ returns a string that contains the switch at the current START
.SWITCHES displays the switch data entered by the client
  ------------------------------------------------------------------------ }

7 CONSTANT #SDIGITS   \ number of digits per switch (doesn't change, but ...)

\ ----- buffer for raw (client) switch data
4096 CONSTANT #SBUF
CREATE SBUF #SBUF /ALLOT  \ Note: must fill buffer with zeroes!
```

ACLM-OK.fs (Cont.)

```
: SWITCHES?  ( -- ? )   SBUF C@ ;  \ true = switches in SBUF

0 VALUE START    \ pointer to start current switch candidate

: /START  ( -- )   SBUF 1+ TO START ;
: +START  ( n -- )   +TO START ;
: NEXT-CHAR  ( -- )   1 +START ;

: #END  ( -- )   SBUF COUNT + ;

: #LEFT  ( -- n )   #END  START - ;  \ # left in switch buffer

: MORE?  ( -- ? )   START #END < ;

: SWITCH$  ( -- a cnt)   START #SDIGITS ;


\ ----- fill switch data buffer
S" switches" GET-CLIENT-CONTENT-VALUE DUP ( text present? )
[IF] SBUF PLACE [ELSE] 2DROP [THEN]

\ S" 2134567456 2343332456666" SBUF PLACE

: .SWITCHES  ( -- )
   BREAK
   SBUF COUNT DUP IF
      S" Switch Data: " BOLD BLUE .<TYPE>
      BOLD BLUE .<TYPE>
   ELSE
      2DROP  S" No Switch Data!" BOLD RED .<TYPE>
   THEN ;
```

ACLM-OK.fs (Cont.)

```
{ ---------------------------------[ Errors ]-----------------------------

EBUF is a buffer that contains all error messages for the current switch
   process.  It is zeroed out so that a zero count signals a processing
   session without errors.
/ERRS initializes the error buffer
!ERR puts a string in the error buffer
!TEXT-ERR puts an error string in the error buffer with a crlf and a visual
   marker to help the user seperate error messages
.ERRS displays the contents of the error buffer with HTML emphasis
  ------------------------------------------------------------------------- }

\ ----- buffer for error messages.  Empty => no errors
128 CONSTANT #EBUF
CREATE EBUF #EBUF ALLOT

: /ERRS  ( -- )   EBUF #EBUF ERASE ;
: !ERR  ( a n -- )   EBUF DUP C@ IF APPEND ELSE PLACE THEN ;
: !TEXT-ERR  ( a n -- )   \ use to start an error message
   <CRLF> COUNT !ERR   S"  | " EBUF APPEND  EBUF APPEND ;
: .ERRS  ( -- )
   BREAK
   EBUF COUNT BEGIN
      $0A SPLIT  2OVER  BOLD RED .<TYPE>
   DUP 2 < UNTIL 2DROP ;

: -ERRS?  ( -- ? )   EBUF C@ 0= ;
```

ACLM-OK.fs (Cont.)

```
{ ----------------------------[ Switch Validation ]-------------------------

?RANGE validates a switch number to see if it is in the District's range.
   Valid switches are displayed and stored in the data buffer for
   later incorporation into a data (deactivation) file.
NUMERIC? validates an ASCII character to see if it is a number
#DIGITS examines #SDIGITS characters starting at the current START and
   returns the number of valid digits.  The number of valid digits
   should be #SDIGITS for a valid switch number (however, this number
   may be outside the District's range -- see ?RANGE above).
?VALID validates the number at the current START position.  Because ?VALID
   is called only after the number is known to be both numeric and of
   the correct length, IS-NUMBER? should never return a FALSE.
   However, a "belt and suspenders" error message is provided for this
   case.
PROCESS does preliminary processing of a switch candidate, shunting short
   switches off for error processing and passing prime candidates on
   to be validated.  It also manages advancing the START pointer to
   the next candidate (via +START) after processing.
VALIDATE is the main processing loop.  After initializing the START
   pointer and the error buffer, it simply scans SBUF a character
   at a time, sloughing non-numeric characters as it goes.  As soon
   as it encounters a numeric character, it assumes it is a switch
   candidate and passes it off to PROCESS for further validation.
   The VALIDATE loop terminates when there are no more characters
   left in SBUF.  The PROCESS and NEXT-CHAR words advance the START
   pointer which is checked by MORE? for loop termination.
VALIDATE-SWITCHES simply checks the result of the switch validation,
   displaying error messages, if necessary.
  ------------------------------------------------------------------------ }

: ?RANGE  ( n -- )   \ range check
   2000000 2250001 WITHIN IF ( in ACLM switch range)
      BREAK S" Valid: " BOLD BLUE .<TYPE>  SWITCH$ BOLD BLUE .<TYPE>
      SWITCH$ !DBUF
   ELSE
      S" Out of Range: " !TEXT-ERR  SWITCH$ !ERR
   THEN ;

: NUMERIC?  ( char -- ? )   [CHAR] 0  [CHAR] 9  1+ WITHIN ;

: #DIGITS  ( -- n)   \ # digits at current adr
   START  #SDIGITS 0 DO
      DUP C@ NUMERIC? NOT IF LEAVE THEN
   1+ LOOP  START - ;
```

ACLM-OK.fs (Cont.)

```
: ?VALID  ( -- )   \ validate number
   START #SDIGITS 0 IS-NUMBER? IF
      ?RANGE
   ELSE
      DROP S" Invalid switch number: " !TEXT-ERR  SWITCH$ !ERR
   THEN ;

: PROCESS  ( -- )   \ process a switch candidate
      #DIGITS #SDIGITS < NOT IF ( length ok)
         ?VALID  #SDIGITS +START
      ELSE
         S" Short: " !TEXT-ERR  START #DIGITS !ERR
         #DIGITS +START
      THEN ;

: VALIDATE  ( -- )   \ Validate switches in buffer
   /ERRS  /START
   BEGIN
      #DIGITS 0> IF PROCESS ELSE NEXT-CHAR ( skip non-numeric) THEN
   MORE? NOT UNTIL ;

: VALIDATE-SWITCHES  ( -- )   \ validates switch format, writes deact file
   .SWITCHES   BREAK .AOK S" Validating Switches..." BOLD .<TYPE>
   VALIDATE -ERRS? IF
      BREAK S" ALL SWITCH DATA ACCEPTED" BOLD BLUE .<TYPE>
   ELSE
      .ERRS
   THEN ;
```

ACLM-OK.fs (Cont.)

```
\ ----------------------[ Password & No Switch Validation ]--------------------

: DEBUG-PASSWORD?  ( a n -- ? )   S" Emases" COMPARE 0= ;
: VALID-PASSWORD?  ( a n -- ? )   S" Gloria" COMPARE 0= ;

: VALIDATE  ( a n -- )
   BREAK  .AOK  S" Validating Password ..." BOLD .<TYPE>
   2DUP VALID-PASSWORD? IF
      SWITCHES? IF
         2DROP ( P/W $)   VALIDATE-SWITCHES
      ELSE
         BREAK S" No Switch Data!" BOLD RED .<TYPE>
      THEN
   ELSE
      BREAK S" Invalid Password: " BOLD RED .<TYPE>  TYPE
   THEN ;

\ ----- clunky way to handle pesky HTML quotes
CREATE PRE-DATA CHAR * STRING <PRE TITLE="Raw HTTP Request">*

: .DEBUG-INFO  ( -- )
   BREAK S" <HR>" TYPE
   BREAK S" <H4>" TYPE  S" DEBUG INFORMATION " TYPE  S" </H4>" TYPE
   BREAK S" Your Cookies: " TYPE  GET-CLIENT-RAW-COOKIE DECODE-HTTP TYPE
   BREAK S" You have accessed this page from " TYPE
      GET-CLIENT-IP TYPE S"  at " TYPE .TS
   BREAK PRE-DATA COUNT TYPE \ Add Tooltip!
   S" The Raw HTTP Request was:" TYPE
      CR SCRIPT-CLIENT USING WEB-CLIENT-RES READ-BUFR LCOUNT TYPE
\   BREAK S" Dated Filename: " TYPE  DATEDFNAME TYPE
   S" <HR>" TYPE   S" </PRE>" TYPE ;
```

ACLM-OK.fs (Cont.)

```
\ ------------------------------[ Main ]-------------------------------------

[DEFINED] GET-HTML-ARGS ( Logix Server Extensions? )
  [IF]
    S" password" GET-CLIENT-CONTENT-VALUE ( -- a n) DUP
    [IF] ( password $ found)
      2DUP DEBUG-PASSWORD? [IF]
        BREAK  .AOK  .( Debug Password: )   TYPE
        .DEBUG-INFO
      [ELSE]
        VALIDATE
      [THEN]
    [ELSE] ( no password)
      2DROP ( bogus $)  BREAK .AOK  .( No Password Entered)
    [THEN]
  [ELSE]
    BREAK S" Error: Server Extensions not defined!" TYPE
  [THEN]
  BREAK .AOK

( ################################ End Forth
################################ ) %>

<STRONG>Finished!</STRONG>

</P>
</BODY>
</HTML>
```

ACLM-SafeOK.fs

```
<HTML>
<HEAD>
<TITLE>ACLM Safety Light Script</TITLE>
<META http-equiv="Pragma" content="no-cache">
<LINK rel="stylesheet" type="text/css" href="Common.css">
</HEAD>
<BODY>

<P>

<% ( ============================== FORTH
================================== )

\ NOTE! This script is called by (and executes after) ACLM-SafeRecvForm.FS

\ ----------------------------[ Client ID  ]-----------------------------

: .AOK  ( -- )   S" ACLM-SafeOK.SHTML -- " TYPE ;
BREAK .AOK .( Executing ...)
```

ACLM-SafeOK.fs (Cont.)

```
{ ------------------[ Switch Buffer, Navigation, Display ]-----------------

SBUF contains the action data entered by the client browser
ACTION$ returns a string that contains the switch at the current START
.ACTIONS displays the switch data entered by the client
  ------------------------------------------------------------------------ }

\ ----- buffer for raw (client) switch data
\ ----- For multiple entries from same Client connection, use only one SBUF!
[DEFINED] SBUF NOT [IF]
   4096 CONSTANT |SBUF|
   CREATE SBUF |SBUF| /ALLOT  \ Note: must fill buffer with zeroes!
[THEN]

: /SBUF  ( -- )   SBUF |SBUF| ERASE ;

: ACTIONS?  ( -- ? )   SBUF C@ ;  \ true = actions in SBUF

\ S" START" SBUF PLACE

: ACTION$  ( -- a n)   \ gets action $, strips leading, trailing blanks
   SBUF COUNT LOCALS| cnt buf |   buf cnt 0 DO DUP C@ 32 = NOT IF LEAVE THEN 1+ LOOP
   DUP buf - ( -- adr' #lead )  cnt SWAP -  ( -- adr' n' )
   DUP 0> IF -TRAILING THEN ;

: .ACTIONS  ( -- )
   BREAK
   ACTIONS? IF
      S" Action Data: " BOLD BLUE .<TYPE>
      ACTION$ BOLD BLUE .<TYPE>
   ELSE
      S" No Action Data!" BOLD RED .<TYPE>
   THEN ;
```

ACLM-SafeOK.fs (Cont.)

```
{ ---------------------------------[ Errors ]-----------------------------

EBUF is a buffer that contains all error messages for the current action
   processing.  It is zeroed out so that a zero count signals a processing
   session without errors.
/ERRS initializes the error buffer
!ERR puts a string in the error buffer
!TEXT-ERR puts an error string in the error buffer with a crlf and a visual
   marker to help the user seperate error messages
.ERRS displays the contents of the error buffer with HTML emphasis
  ------------------------------------------------------------------------- }

\ ----- buffer for error messages.  Empty => no errors
\ ----- again, use only one instance of buffer for each Client session
[DEFINED] EBUF NOT [IF]
   128 CONSTANT |EBUF|
   CREATE EBUF |EBUF| ALLOT
[THEN]

: /ERRS  ( -- )   EBUF |EBUF| ERASE ;

: !ERR  ( a n -- )
   EBUF DUP C@ IF APPEND ELSE PLACE THEN ;

: !TEXT-ERR  ( a n -- )   \ use to start an error message
   <CRLF> COUNT !ERR   S"  | " EBUF APPEND  EBUF APPEND ;

: .ERRS  ( -- )
   BREAK
   EBUF COUNT BEGIN
      $0A SPLIT  2OVER  BOLD RED .<TYPE>
   DUP 2 > NOT UNTIL 2DROP ;

: -ERRS?  ( -- ? )   EBUF C@ 0= ;
```

ACLM-SafeOK.fs (Cont.)

```
{ ----------------------[ Action Execution and Validation ]-----------------

WRITE-FLAG  given the name of a flag file, writes that file
VALIDATE-ACTIONS is the main action processing loop.  It calls
   VALIDATE-BUFFER to validate the entered string and handles error and
   status display.
VALIDATE-BUFFER checks the entered action string, displaying
   error messages as necessary.
  ------------------------------------------------------------------------- }

: WRITE-FLAG  ( a n -- )   \ Write flag file
   LOCALS| cnt adr |
   adr cnt R/W CREATE-FILE ?DUP IF ( Couldn't)
      NIP ( fid)  (THROW) BREAK .ERR$
      BREAK adr cnt .ERR$
   ELSE
      ( fid)  CLOSE-FILE
   THEN ;

: .note  S" Note: " !TEXT-ERR ;
: .error  S" Error: " !TEXT-ERR ;

: VALIDATE-BUFFER  ( -- )   \ Validate actions in buffer
   /ERRS
   ACTION$ S" START" COMPARE(NC) 0= IF
\      BREAK S" Found Start!" BOLD RED .<TYPE>
      S" C:\ACLM\SAFE\SAFE.flg" WRITE-FLAG
   ELSE
      ACTION$ S" STOP" COMPARE(NC) 0= IF
\         BREAK S" Found Stop!" BOLD RED .<TYPE>
         S" C:\ACLM\SAFE\STOP.flg" WRITE-FLAG
      ELSE
         .error  S" No action match" !ERR
         .error  S" Action must be either START or STOP" !ERR
         .note   S" Action is not case sensitive" !ERR
         .note   S" Leading spaces are ignored" !ERR
      THEN
   THEN ;
```

ACLM-SafeOK.fs (Cont.)

```
: VALIDATE-ACTIONS  ( -- )   \ validates switch format, writes deact file
   .ACTIONS   BREAK .AOK S" Validating Actions..." BOLD .<TYPE>
   VALIDATE-BUFFER -ERRS? IF
      BREAK S" ACTION ACCEPTED!" BOLD BLUE .<TYPE>
   ELSE
      .ERRS
   THEN ;


\ ----------------------[ Password & No Switch Validation ]--------------------

: DEBUG-PASSWORD?  ( a n -- ? )   S" Emases" COMPARE 0= ;
: VALID-SAFETY-PASSWORD?  ( a n -- ? )   S" Safe1" COMPARE 0= ;


\ ----- clunky way to handle pesky HTML quotes
CREATE PRE-DATA CHAR * STRING <PRE TITLE="Raw HTTP Request">*

: .DEBUG-INFO  ( -- )
   BREAK S" <HR>" TYPE
   BREAK S" <H4>" TYPE  S" DEBUG INFORMATION " TYPE  S" </H4>" TYPE
   BREAK S" Your Cookies: " TYPE  GET-CLIENT-RAW-COOKIE DECODE-HTTP TYPE
   BREAK S" You have accessed this page from " TYPE
      GET-CLIENT-IP TYPE S"  at " TYPE .TS
   BREAK PRE-DATA COUNT TYPE \ Add Tooltip!
   S" The Raw HTTP Request was:" TYPE
      CR SCRIPT-CLIENT USING WEB-CLIENT-RES READ-BUFR LCOUNT TYPE
\   BREAK S" Dated Filename: " TYPE  DATEDFNAME TYPE
   S" <HR>" TYPE   S" </PRE>" TYPE ;
```

ACLM-SafeOK.fs (Cont.)

```
\ ------------------------------[ Main ]-------------------------------------

[DEFINED] GET-HTML-ARGS ( Logix Server Extensions? )
  [IF]
    S" password" GET-CLIENT-CONTENT-VALUE ( -- a n) DUP
    [IF] ( password $ found)
      BREAK .AOK  .( Validating Password ...)
      2DUP DEBUG-PASSWORD? [IF]
        BREAK  .AOK  .( Debug Password: )   TYPE
        .DEBUG-INFO
      [ELSE]
        2DUP VALID-SAFETY-PASSWORD? [IF]
          2DROP  ( safety P/W)
          /SBUF  ( just in case ...)
          S" actions" GET-CLIENT-CONTENT-VALUE DUP ( text present?)
          [IF]
            SBUF PLACE VALIDATE-ACTIONS
          [ELSE]
            2DROP BREAK .( No Action Data!)
          [THEN]
        [ELSE]
          BREAK .( Invalid Safety Password: ) TYPE
        [THEN]
      [THEN]
    [ELSE] ( no password)
      2DROP ( bogus $)  BREAK .AOK  .( No Password Entered)
    [THEN]
  [ELSE]
    BREAK S" Error: Server Extensions not defined!" TYPE
  [THEN]
  BREAK .AOK

( ################################ End Forth
################################ ) %>

<STRONG>Finished!</STRONG>

</P>
</BODY>
</HTML>
```

Common.css

```
/* Logix Cascading Style Sheet. (c)2002 Logix */

body  { font-size: 75%;
   line-height: 125%;
   font-family: Verdana, Arial, Helvetica,;
   color: black;
   background: white; }

em { color: #004080;
   font-size: 100%;
   font-style: normal;
   font-weight: italic; }

strong  { color: #004080;
   font-size: 105%;
   font-style: normal;
   font-weight: bold; }

a:link    { color: #0000FF; }

a:active  { color: #FF33CC; }

a:visited { color: #800080; }

a:hover   { color: #FF0000; }

a:sidebar { color: #339900; }

a:visited.popover  { color: #0000A0;
   text-decoration: none; }

a:link.popover  { color: #0000A0;
   text-decoration: none; }

a:hover.popover  { color: #000060;
   text-decoration: none; }
```

Common.css (Cont.)

```
h1 { font-size: 155%;
   margin-bottom: 1em;
   line-height: 125%; }

h2 { font-size: 135%;
   margin-top: 1.5em;
   margin-bottom: .5em; }

h3 { font-size: 115%;
   margin-top: 1.2em;
   margin-bottom: .5em; }

h4 { font-size: 100%;
   font-weight: bold;
   color: #008000;
   margin-top: 1.2em;
   margin-bottom: .0em; }

h5 { font-size: 100%;
   font-weight: bold;
   margin-top: 1.2em;
   margin-bottom: .0em; }

p  { margin-top:6pt; margin-bottom: 10pt; }

p.footer { font-size: 85%;
   color: navy;
   margin-top: 2em;
   margin-bottom: .5em; }

li p { margin-top: .6em;
   margin-bottom: 0em; }

big  { font-weight: bold;
   font-size: 105%; }
```

Common.css (Cont.)

```
ol { margin-top: .5em;
   margin-bottom: 0em; }

ul { margin-top: .6em;
   margin-bottom: 0em;
   margin-left: 2.75em; }

ol ul { list-style: disc; margin-top: 2em; }

li { padding-bottom: .3em;
   margin-left: -1.25em; }

dl ul { margin-top: 2em;
   margin-bottom: 0em; }  /*list item inside a def/term*/

dl { margin-top: -1em; }

ol dl { margin-top: -1.5em;
   margin-left: 0em; }      /*term/def list inside a numbered list*/

ol dl dl  { margin-top: 0em;
   margin-left: .2em; }      /*term/def list inside a term/def list*/

dd { margin-bottom: 0em;   /*not currently working*/
   margin-left: 1.5em; }

dt { padding-top: 2em;
   font-weight: bold;
   margin-left: 1.5em; }

code { font-family: Courier; }

pre  { margin-top: 0em;
   margin-bottom: 1.5em;
   font-family: Courier; }
```

Common.css (Cont.)

```
table { font-size: 100%;
   text-align: left; }

tr { margin: .50em;
   vertical-align: top;
   }

th { text-align: left;
   margin: .50em;
   vertical-align: top;
   background: #D0D0FF; }

td { margin: .50em;
   vertical-align: top; }

.ChanAssign  { font-size: 8pt;
   text-align: left;
   color: black;
   background: white
   font-family: Andale Mono; }
```

Bad-URL.fs

```
<HTML>
<HEAD>
<TITLE>Page Not Found - Unknown URL (404)</TITLE>
</HEAD>
<BODY>
<H2>There was a problem with your request, sorry ...</H2>
<% ( We're in FORTH! )
   PARA
   S" The page you requested " .<TYPE>
   .( <CODE>) SCRIPT-CLIENT USING WEB-CLIENT-RES GET-RAW-READ
   BL PARSE-WORD 2DROP BL PARSE-WORD 2NIP DKRED-COLOR COLOR .<TYPE> .( </CODE>)
   S"  does not exist." .<TYPE>
   END-PARA
   PARA .( Please correct your request and try again.)
   END-PARA
   HTML.SERVER
%>
</BODY>
</HTML>
```