CorreLog®

Sigma Web Framework

User And Developer Reference Manual

http://www.correlog.com mailto:info@correlog.com

CorreLog Sigma Web Framework, User And Developer Reference Manual

Table of Contents

Section 1: Introduction

This document provides a detailed description of the CorreLog Sigma Web Framework, which is a comprehensive, flexible, and lightweight system for creating web sites and web-based applications. In particular, this framework forms the basis for the CorreLog Security and Log Monitor Server, documented in the "CorreLog User Reference Manual."

The CorreLog Sigma Web Framework shares similarities with a variety of other frameworks, but is designed to be extraordinarily lightweight, simple to use, and extensible.

This manual is intended for use by web administrators, responsible for installing and maintaining the CorreLog system, developers who intend to use the CorreLog system to develop web based systems, and also end-users who will be using the CorreLog system to access web based information.

Note that this manual covers only the CorreLog Sigma Web Framework, and does not address individual applications that may possibly be running within the CorreLog system. In particular, specific information on the CorreLog Server, and other applications can be found in the appropriate user manuals addressing these individual components.

Why Another Web Framework?

A programming framework is an implementation of a system that supports rapid development of usable software for a specific purpose, typically including libraries, utilities, services, programs, and methodologies. The term “Framework” is also often used to describe programming languages, methodologies and Interactive Development Environments IDEs), although this may be too general a usage of the term Framework, which is often used inappropriately as a “buzz word”.

Sigma is a web-programming framework. Additionally, Sigma may also refer to customized applications that run within the Sigma framework. These Sigma applications include the CorreLog Syslog Agent, Sigma RSS Aggregator, Sigma DEX Data Extraction Engine, Sigma SLED Correlation Engine, and Sigma Reporting facility. These particular Sigma applications all rely on the Sigma Web Framework, which is the exclusive topic of this manual. (For information on a specific Sigma application, including those above, refer to the appropriate application manual.) Unless otherwise specified here, the term “Sigma” will refer to the “CorreLog Sigma Web Framework” described in this manual.

Web-programming frameworks proliferate. There are many good web-programming frameworks already in existence, including “Ruby on Rails”, “CakePHP”, “and Symfony”, and many others. This begs the question: Why yet another web framework? Given that many different frameworks have come into existence, become popular, and then faded from the scene, is it worthwhile to spend the effort to learn and implement yet another framework system?

The answer is simple: Sigma is intended to be extraordinarily simple, quick to install, and flexible. It is an ultra-light weight, flexible framework that operates as a simple extensible web shell that is independent of any particular programming language or methodology. It does not enforce any particular programming language, Sigma does not include an IDE which has to be learned by a programmer, and does not require any substantial learning curve. As the Greek letter “Sigma” implies, this framework is designed to act as a “summing” program that permits often-trivial integration of various web-based applications. Considering that Sigma can be up and running almost instantly, the return on investment in using Sigma is very high.

Sigma is intended as a framework that promotes an exceptionally long life cycle, since it works with new and old software systems. It is designed to be learned quickly, and allow easy customization and extensibility that greatly prolongs its usability and life span in the face of rapidly changing fads and styles of software development.

Sigma Web Framework Features

Some of the specific features of Sigma are discussed below:

- **Automatic generation of nested tabs.** One of the most characteristic elements of the Sigma framework is its function of automatically creating nested navigation tabs. This permits easy creation and navigation of a complex applications and multiple.

- **HTTP authentication management**. Sigma incorporates read-to-run HTTP authentication as a standard feature. This permits selective access to various areas of the website based upon admin, user, and guest privileges.

- **User login management**. Sigma includes a user login management interface, which allows an administrator to create user logins and permission sets via the web interface. Developers can easily limit access to certain applications based upon the role of the user.

- **CGI interface support.** In addition to serving simple HTML and JavaScript pages, Sigma allows a developer to easily create dynamic applications in a variety of programming language, including C, C++, Java, PHP, Perl, Ruby, VB, and others. Unlike many frameworks, which are dependent on particular programming languages and methodologies, Sigma is independent of these.

- **Support for Windows batch file programs.** Although Sigma is ideal as a framework for professional programmers, the system also supports creation of dynamic CGI screens using trivial Windows Batch files. (This demonstrates how easy it is to add new screens to Sigma using minimal programming power.)

- **Windows Service Manager interface.** Sigma incorporates an interface to the Window Service Manager, which allows developers to run programs as background processes, and as standard windows services. This is accomplished via configuration files, and requires no programming of the Windows SCM or registry updates.

- **Built in scheduler program**. Sigma provides a built in scheduler program, that can be used to automatically run programs on system startup, shutdown, and ad certain intervals. This provides one way for developers to create dynamic web pages (which are constructed as background processes at periodic intervals.)

- **Easily customizable features.** Sigma provides various “signature” elements, such as its navigation tables. However, Sigma also incorporates

many customizable features that allow a developer to change screen colors, screen headers and footers, and modify the look and feel of the user experience.

- **Ready-to-run Apache server**. As part of the Sigma framework, this system incorporates a pre-configured version of the industry standard Apache HTTP server, which has been tuned to provide fast, simple, and secure web services with minimal intrusion on CPU, memory, or disk space.

- **Support for Microsoft IIS.** Sigma additionally supports the IIS server, including support for authentication using membership in Windows user groups, to simplify the assignment of permissions. Sigma can also be easily integrated with other third-party and specialty HTTP servers, including Apache Tomcat.

- **C / C++, other API services.** Sigma works with any programming language and methodology, and furnishes libraries, examples, and templates, and documentation to permit developers to rapidly create CGI programs and screens, including macro based web pages.

The One-Minute Introduction To Sigma

The remainder of this manual will deal with the various detailed aspects of the Sigma server, including operation, administration, development, and customization. For those users wishing to immediately assess what Sigma does, all important Sigma information is contained below.

- After installing Sigma, you can access the Sigma server via a web browser at port 80 or port 88, or port 888 or port 8888. (The particular port number is displayed and configured in the installation dialog, and is by default port 80, but may also default to one of the other port numbers if port 80 is busy.) You can typically access Sigma via a desktop shortcut on the machine where Sigma is installed, or via the Windows “Start” menu. (See section on “Installation” in this manual for more information.

- The default login to Sigma is username “admin”, password “admin”. After logging into the system, you can click on the tabs, at the top of the browser display, selecting first “System” and then “Logins”. This will permit you to view and modify the login username or password, or add other user logins. (See the section on “Login Management”, in this manual, for more information.)

- Note the signature characteristic of “Sigma”, which is a system of nested tabs at the top of the display. You click on these tabs to access

applications. The precise nesting order of tabs is strictly defined by the directory structure residing in the very important “sigma-web” directory of the system.

- To create new tabs, you can either create files within “sigma-web”, or create new directories. The naming convention of these files and directories is somewhat specific. Each file or directory name must begin with a three-digit integer number and underscore. Each file or directory name must end with a specific supported suffix. This yields file and directory names such as “100_Welcome.html” or “900_System.dir”. (See the section on “Adding Web Pages”, in this manual for more information.)

As described above, to add a new web page to the system, simply copy the web page into the “sigma-web” folder of the system, using the specific naming conventions described above (and in detail in later sections.) This will create a tab on the web browser display. If you create a directory, you can then copy files and more directories into that directory in order to create nesting of applications.

Finally, the “sigma-web” directory can contain either files or executables. In particular, the “sigma-web” directory can contain programs, scripts, and batch files. For example, you can create a very simple “000_Hello.bat” file and copy this file into the “sigma-web” directory. This will add a “Hello” tab to the display. When a user clicks on this tab, the server executes the batch file and displays the resulting standard output of the batch file.

Further sections will describe in detail the various other features, adaptations, customizations, and applications associated with the Sigma system. The reader is encouraged to experiment with the system. In particular, almost all of the information required to understand the essentials of the Sigma Web Framework has now been explained; so feel free to get started!

Section 2: Framework Installation

The Sigma Web Framework is usually delivered as a self-extracting WinZip file, and contains install and uninstall programs, residing in the “system” directory of the Sigma root directory. The install program normally starts after files are extracted. To uninstall the system, the operator access the Windows “Add / Remove” programs application, and clicks on the “Sigma Framework” entry.

Sigma is specifically designed not to scatter DLL or other files into system directories. All files reside in the Sigma root directory, by default the directory C:\Sigma, or C:\CorreLog (although this directory may be specified differently when extracting files.) Therefore, if the user stops the Sigma Framework service, the entire Sigma directory can be simply dragged and dropped into the Windows “Recycle Bin” and this will effectively discard the entire installation. (However, note that this will still leave the service entry for Sigma, within the Windows Service Manager, which is normally cleaned up by the Uninstall procedure.

The Sigma install and uninstall programs are very simple, and require very little explanation. This section describes the detailed steps needed to install and uninstall the system, as well as containing application notes that may be useful to system developers and managers.

System Installation Requirements

The Sigma Framework is minimally invasive, and can be installed on a variety of platforms and operating systems. An “Administrator” login is required to install the software. Specific system requirements of the Sigma Framework system are described below.

- **Disk Space.** The Sigma Framework, by itself, has a small footprint of less than 10 Mbytes, but the actual disk space may vary depending upon the particular applications installed (or which might later be installed) as part of the framework.

- **Operating System.** The system can be installed on Windows XP, Vista, as well as Windows NT 2000 systems. Sigma, by itself, does not require Java, or .NET to be installed on the platform.

- **CPU Requirements**. The Sigma Framework makes minimal use of CPU, and can co-exist with other server components and applications. The actual CPU required by the program depends mainly on the CPU load of the HTTP server, which depends upon how often the server is accessed and by how many users. Typically, on a normal machine, the CPU usage of the program will be less than 10%.

- **TCP Connectivity.** The Sigma Framework is a web server and web based application. It cannot be installed on a platform that does not have TCP connectivity. (This will typically not be a problem, but may occur in certain evaluation and test scenarios.) The TCP software should be a standard installation. There are no special requirements associated with the Network Interface Card (NIC). Although Sigma can work in a DHCP environment, for best results the host platform should have a permanent IP address, and DNS services.

- **Service Ports.** The Sigma Framework web server requires control of a single TCP service port (normally port 80, but possibly any other port selected by the user.) The installer attempts to auto-detect a free service port. Port blocking and virus protection programs (in particular McAfee Virus Scan) may interfere with this. (See notes below.)

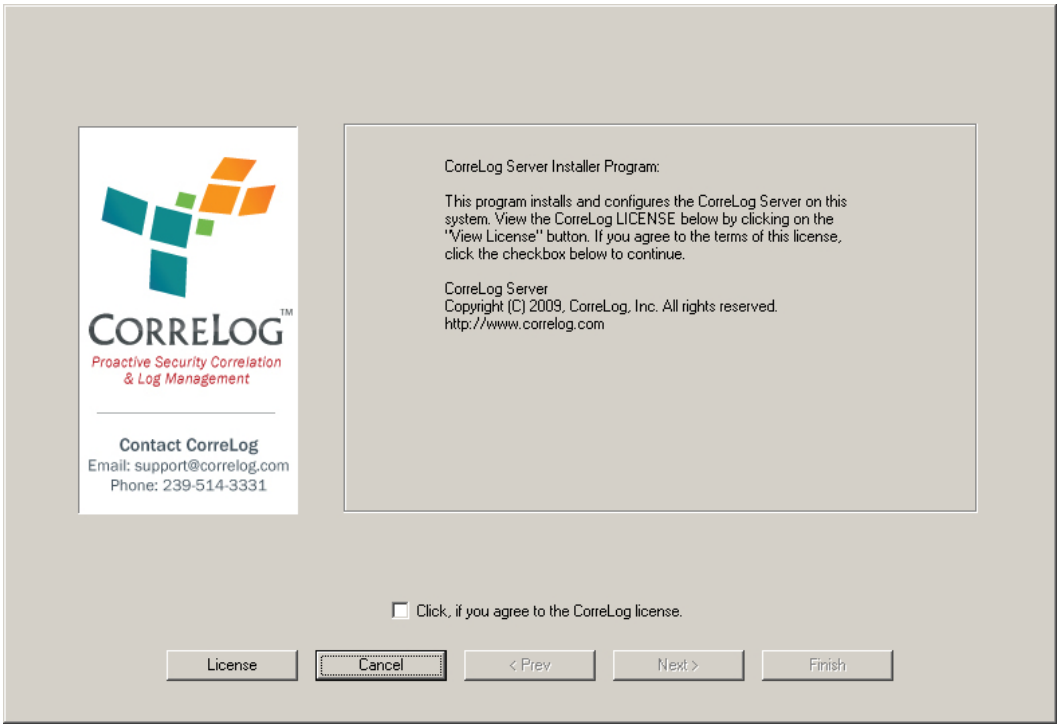
To insure proper installation of the program, the user should close all windows, and temporarily disable any port blocking or Virus Scan software on the system. Any errors, detected during the installation process, stop further progress of the installation program with an error dialog that should be addressed prior to the continued execution of the program. Reboot, after installation, is not required, but is recommended.

Executing the CO-install.exe Program

The name of the Sigma Web Framework installation program is “CO-install.exe”, and this program is located in the “system” directory of the Sigma root directory. This program normally is executed automatically, after extracting files from the delivery package.

The “CO-install.exe” program may also be executed at any time by changing working directories to the Sigma directory, and executing the program manually. This may be necessary to reconfigure the Apache server, or to change the installation location to some other disk or directory, as discussed in a later section.

In particular, the CO-install.exe program can be run multiple times, and will perform a quick uninstall and re-install of any system components. A depiction of the CO-install.exe program installation dialog is shown below.



As shown above, the installation dialog is a standard Windows dialog containing Prev, Next, Cancel, and Finish buttons. The user can exit the dialog at any time by clicking the “Cancel” button.

The dialog provides four screens, as follows:

1. The initial screen provides an introduction. Before the user can click the “Next” button, then should view the license file, and then click the

checkbox to agree to the terms of the license. This enables the “Next” button.

2. The second screen queries the user for the HTTP service port to use with the Sigma Apache server. Even if the user elects not to use the Apache server (as discussed in the next section) the user should specify a port number, and then click “Next”.

   Note: The installer will attempt to select a free service port, checking first port 80, then port 88, then port 888, then port 8888. This suggested service port is displayed in the edit box on the second screen, and can be modified by the user prior to clicking the “Next” button. (Also, see notes following the next step.)

3. The third screen checks the service port, entered on the second screen. If the service port is available, the user is directed to finish the installation procedure. If the service port is not available, an error popup is generated, and the user is directed to select another service port.

   Note: Virus protection programs, and port blocking software can affect the test of the service port availability. In particular, the popular McAfee Virus Scan program may cause the specified TCP port to appear free when it is actually busy. The user should verify that the port is actually free, such as by executing “netstat –a” at a command prompt.

4. After clicking “Next” on the third screen, the actual installation procedure is performed. This configures the Apache HTTP server, registers the Sigma Framework service with the Windows Service Manager, creates the “Add/Remove Programs” entry for the system, and adds a desktop shortcut. The success or failure of the program is indicated on the final screen of the dialog.

After running the above install procedure, the user can click the “Sigma” desktop shortcut and login to the Sigma Framework. The precise URL and default login to the system are displayed on the last screen of the installation dialog, and the user should make note of these values. By default, the username to sigma is “admin”, password “admin”. If security is a concern, this value should be changed immediately, as discussed in Section 3 of this manual.

Selecting An HTTP Server Port

The second screen of the installation dialog prompts the user to select an HTTP server port. The user can select any free port on the system. The installation dialog performs a quick check of likely service ports, and suggests (in the following order) Port 80, Port 88, Port 888, and Port 8888. The installer can accept the suggested value, or can select any other value before clicking “Next”

on the second screen. The install program will validate any TCP port number entered by the user, before displaying the third screen.

Virus Protection and Port Blocking software can interfere with the auto-detection and suggested value. In this case, the install program may suggest or accept a service port number that is actually busy. If this happens, the HTTP server will fail to start, and the install dialog will have to be re-executed.

If the user experiences this problem, it may be helpful to inspect the current TCP service ports being used on the system. This is accomplished by executing the Windows “netstat –a” command, at a Windows command prompt. This utility lists all the service ports that are currently busy on the system, and may be useful in determining a particular service port to specify in the dialog.

Using Sigma With IIS Or Third Party HTTP Servers

The Sigma Framework does not rely on any special features of the Apache server. The Sigma Framework can be used with other HTTP servers, in particular Microsoft IIS server.

In order to configure a third party server, the user creates two virtual directories (i.e. Virtual Web Folders), as follows:

- **/s-cgi** – This web folder corresponds to the “Sigma\s-cg” directory of the framework. The user must grant this web folder both read and execute permissions.

- **/s-html** – This web folder corresponds to the “Sigma\s-html” directory of the framework. The user can grant this web folder read-only permissions.

If the Sigma Administrator wishes to make use of IIS authentication (possibly to take advantage of Windows Directory Services), the username and password of any registered user must appear in the Sigma “Login” registry. (See later sections on how to configure user logins and other associated parameters for more information. (Note that this is required to take advantage of the permission features of the Sigma Framework.

After configuring IIS to work with the Sigma Framework, the user can disable the Apache server, or leave it running as an auxiliary interface. To disable the Apache service, the Sigma administrator edits the “Schedule” tab of Sigma, as discussed in a later section. The user can additionally simply remove or rename the “Sigma\apache\CO-apache.exe” file to disable the startup of this program.

Changing The Location Of The Sigma Root Directory

The “CO-install.exe” program, in addition to installing and configuring Sigma, can also be used to move the location of the Sigma root directory after installation. To accomplish this, use the following steps:

1. Stop the existing Sigma installation using the Windows Service Manager, or by executing the “net stop sigma” command at a command prompt. Verify using the Windows Task Manager that all “CO-“ processes have been stopped.

2. Copy the Sigma root directory to its new location using a normal folder copy command.

3. Change working directories to the new “Sigma\system” directory, or navigate to this folder using Windows File Explorer.

4. Execute the “CO-install.exe” program manually, and follow the steps described earlier in this section, needed to re-install the service and reconfigure the Apache HTTP server.

5. Optionally, copy the “Sigma\apache\passwords.dat” file from the old location to the new location, in order to preserve the password database.

   Note: The installation procedure rewrites the Apache password file in order to provide the default login to the system. The user can also login to the system using the default “admin” password “admin” login, access the Sigma login editor, and resave the configuration data in order to update the passwords.dat file with current settings.

6. Optionally, remove the old Sigma directory.

If the site is implementing IIS or some other third-party HTTP server, additional steps will be required to reconfigure that server. No other steps are needed to modify the location of the Sigma installation.

Note that applications, running in the framework, may have their own procedures, required as part of any relocation process. Refer to the specific documentation associated with a Sigma Framework application for further notes.

Uninstalling The Sigma Framework

The Sigma Framework is uninstalled via the “Add / Remove Programs” windows facility. The user navigates to this screen (via the Control Panel) and clicks on

the “Sigma Framework” entry to execute the Uninstall program. The user follows the instructions of the dialog to uninstall the Sigma Framework system.

Note that, unlike most uninstall programs, the Sigma Framework files are left intact on the disk. Following the uninstall procedure, the user must physically remove these files, such as by dragging the Sigma root directory to the Microsoft Windows “Recycle Bin”. This extra step safeguards any accidental removal of data on the system.

After running the uninstall procedure, but before manually removing the Sigma files manually, an administrator can re-execute the Sigma Installer program to re-install the registry keys other Sigma configuration components, as described previously.

Re-running the CO-install.exe program after uninstalling the Sigma framework will re-initialize the Apache password file, but will otherwise have no detrimental effect on the system. The administrator will be able to restore the password database by logging into the Sigma system, accessing the “System > Logins” screen, and regenerating the passwords by editing any entry or adding a new user.

In addition using the “Add / Remove Programs” facility, the user can execute the “Sigma\system\CO-uninst.exe” program manually. This has an identical effect to launching this program via the standard Windows “Add / Remove Programs” application.

Extending The Install and Uninstall Programs

The CO-install.exe and CO-uninst.exe programs have several additional features that permit them to launch other install and uninstall programs as part of their normal innovation. This may be useful in packaging the Sigma Framework with third party applications.

In particular, the CO-install.exe program will attempt to execute any “CO-install.bat” file found in the “Sigma\system” directory. This occurs as a final step in the program’s execution. Standard output of this batch file will be viewed as part of the CO-install transcript.

Likewise, the CO-uninst.exe program will execute any “CO-uninst.bat” file found in the “Sigma\system” directory when the user clicks on the “Finish” button. (The CO-uninst.bat file is launched as a background file, after the SG_uninst.exe program exits.) This permits a simple way of extending these programs to encompass larger requirements.

Section 3: Framework Screens

The Sigma Framework comes with several built-in screens, discussed in this section. These screens permit the user to modify logins, modify system parameters, and schedule the execution of background programs on the system.

- **Login Screen.** This screen allows an administrator to add, edit and delete system logins. The screen is used both by the native Apache HTTP server, but also required if the administrator elects to run Sigma with a third-party HTTP server, such as Microsoft IIS.

- **ODBC Screen.** This screen assists in interfacing the Sigma system to an external SQL database using the Microsoft "Open Database Connectivity" standard.

- **Parms Screen**. This screen allows an administrator to modify specific performance parameters of the framework, including colors, timeouts, and operating modes.

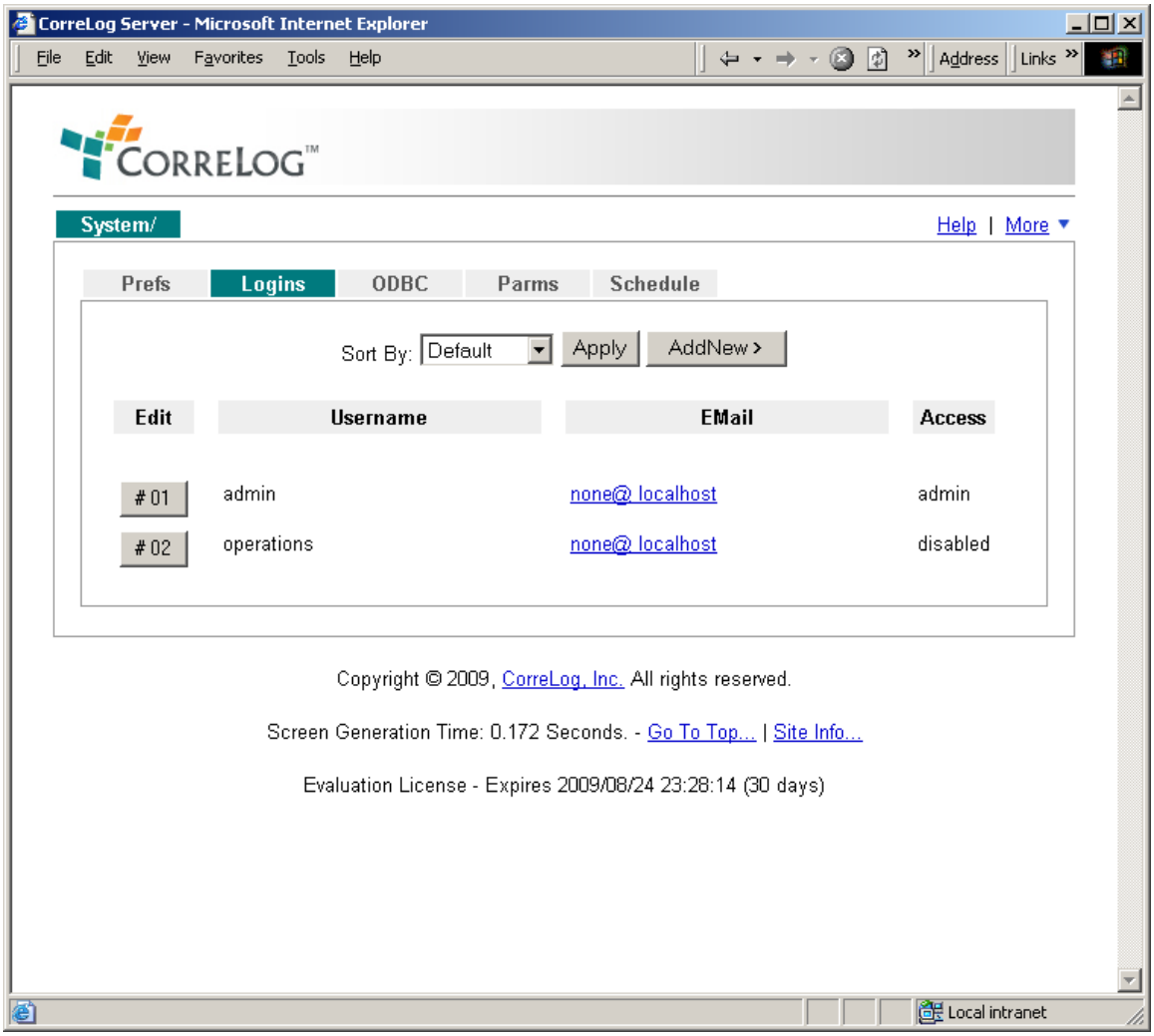
- **Schedule Screen.** This screen allows an administrator to configure the built in scheduler of Sigma, which permits programs to be launched on system startup and shutdown, as well as launched at periodic intervals.

The above screens are a part of all Sigma implementations, regardless of the applications and screens that may also exist. Each screen is detailed in the sections that follow.

The Sigma “Login” Screen

The Sigma Framework Login Screen allows an administrator to configure usernames and passwords and access levels to the program. This screen is accessed after login by clicking on the “System” tab, and then clicking the “Login” tab (as needed.) The “Login” screen is available ONLY to users who have been given “admin” rights to the system. Otherwise, attempting to access this screen will result in a “Permission Denied” screen.

The “Login” screen is depicted below.



The “Login” screen is a standard Sigma dialog. To add a new login entry to the system, the user clicks on the “AddNew” button. To edit an existing login entry, the user clicks the “Edit” button, followed by the “Commit” button on the edit screen. To delete an existing login entry, the user clicks the “Edit” button, followed by the “Delete” button on the edit screen. Selecting an option from the

“Sort By” dropdown menu, and then clicking “Apply” can sort screen fields. By default, logins are sorted in the order in which they were added to the system.

As part of each login, the administrator specifies an “Access” level of “admin”, “user”, “guest”, or “disabled”. These settings affect the particular login, limiting access to the Sigma framework as follows:

- **Admin Access.** Administrators are permitted access to all screens of Sigma, including the “Login” screen above. No checks are applied to logins with “Admin” access.

- **User Access.** Users are permitted access to most screens, except for any screen in the “System” tab, such as the screens in this section. Users may also have limited access to specific Sigma Framework applications, as determined by the application designer. (Refer to specific applications manual for more details.) Generally Users are permitted to edit configuration data.

- **Guest Access.** Guests are permitted to access most screens, but are not permitted to edit or change any configuration data on the system. Guests cannot access “Edit” or “AddNew” buttons on any screen. (Application designers may override this. Refer to specific application manuals for more details.)

- **Disabled Access.** Disabled users are not permitted access to Sigma. Attempts to access any screen within Sigma is immediately stopped. This has important implications for security, and works with the “Default User” parameter setting discussed in a later section.

Login Screen Operation

Logins are stored in encrypted format within the “config/pass.cnf” file of the Sigma Framework, and are transferred to the “apache/passwords.dat” file (also encrypted) whenever any edit is applied to the “config/pass.cnf” file. Therefore, if the “apache/passwords.dat” file is deleted or corrupt, an administrator can simply access the “Login” screen, access any login, and commit the login in order to completely generate the “apache/passwords.dat” file from scratch.

If the administrator has elected to install IIS, and is using Microsoft Directory Services, then the password values are ignored. However, the Sigma username must precisely match the IIS username in order to correctly apply permissions to the system. A default user access can be established using the “Parms” screen discussed in the next section.

The Sigma "ODBC" Screen

The Sigma Framework "ODBC" Screen allows an administrator to configure usernames and passwords and other parameters associated with an SQL database, using the Microsoft Open Database Connectivity facility. The screen provides general utility in running queries on an arbitrary database, and is mainly useful for program development and system troubleshooting. This screen is accessed after login by clicking on the “System” tab, and then clicking the “ODBC” tab. This screen is available ONLY to users who have been given “admin” rights to the system. Otherwise, attempting to access this screen will result in a “Permission Denied” screen.

The “ODBC” screen is depicted below.

The "ODBC" screen is not required to interface Sigma to a database. (This function is mainly performed via the "RunSQL.exe" program, in conjunction with the Sigma "Macro" capability, both discussed in Appendix B.) However, the

screen is highly useful for generating quick SQL queries, creating tables, performing system maintenance operations, and debugging the system.

Prior to using the ODBC screen, the administrator should configure an ODBC "System DSN" via the Control Panel > Admin Tools > Data Sources" dialog of the Windows system. The data source will then appear in the pull-down select menu of the screen.

When the data source is first selected, the user must click the "Edit" button on the screen to configure the Sigma side of the data source. This displays the "Edit Data Source" screen, which provides the following parameters.

- **Database User Name.** This is the username required to access the data source (if any) and the value must agree with that configured in Windows. If the database does not require a username, the value is "None".

- **Database Password.** This is the password required to access the data source (if any) and the value must agree with that configured in Windows. If the database does not require a password, the value is "None".

- **Use Database.** If there is a sub-database within the main database, it can be configured here (which will simplify queries). This value will prefix database tables and columns. If the database does not contain any sub-databases, the value is "None".

- **Use as Default DSN.** This value is either "Yes" or "No", and controls the initial database that is displayed when the screen is first entered into by the user. Setting the value to "Yes" causes the named database to be displayed in the top-level select menu of the screen on user entry. Setting this value to "Yes" also sets the value to "No" for any other DSN on the system.

The screen works ONLY with "System DSN" names, and does not work with "User DSN" names. Any database that supports ODBC can be specified as part of the Windows "Admin Tools > Data Sources" screen, including Oracle, MS-SQL, and MS-Access.

In particular, the user can configure an MS-Access database on any system, without requiring any other database to be installed. The MS-Access driver is a part of all Windows systems, and does not require Microsoft Office or any special software to be installed other than the native Windows operating system.

The "ODBC" screen is particularly useful in configuring and testing other aspects of the Sigma SQL support features. Detailed notes on using the Sigma SQL interface are provided in Appendix B of this document.

The Sigma “Parms” Screen

The Sigma Framework Parms screen allows an administrator to configure various parameters for the Sigma Framework. This includes a standard list of parameter values, as well as possibly other parameters added by users. This screen is accessed after login by clicking on the “System” tab, and then clicking the “Parms” tab (as needed.) The “Parms” screen is available ONLY to users who have been given “admin” rights to the system. Otherwise, attempting to access this screen will result in a “Permission Denied” screen.

The “Parms” screen is depicted below.

The “Parms” screen is a standard Sigma dialog. To edit parameter entries, the user clicks the “Edit” button, modifies the parameter, and then clicks the “Commit” button on the edit screen. The user can restore the preset default configuration by clicking on the “Default” button on the edit screen. If the “Default”

button is accidentally clicked, the user can inspect the previous settings by clicking the “Back” button of the browser.

The following parameters are supported. (Other parameters may also exist, depending upon the Framework applications that are supported.)

- **CGI Timeout Seconds.** This is the maximum time, in seconds, which the system will wait when generating a screen before displaying a Timeout indication. This setting is useful in preventing errant applications from permanently hanging on the system.

- **Nav Tab On Color.** This is a standard HTML color, or hexadecimal code, indicating the color of navigation tabs that are currently selected on the system. The value can be a standard color (such as “blue”) or a hexadecimal value in standard RRGGBB format.

- **Nav Tab On Text.** This is a standard HTML color, or hexadecimal code, indicating the color of selected navigation tab text, which should be selected to contrast with the Nav Tab On Color, described above.

- **Nav Tab Off Color.** This is a standard HTML color, or hexadecimal code, indicating the color of navigation tabs that are not selected on the system. The value can be a standard color (such as “blue”) or a hexadecimal value in standard RRGGBB format.

- **Nav Tab Off Text.** This is a standard HTML color, or hexadecimal code, indicating the color of selected navigation tab text, which should be selected to contrast with the Nav Tab Off Color, described above.

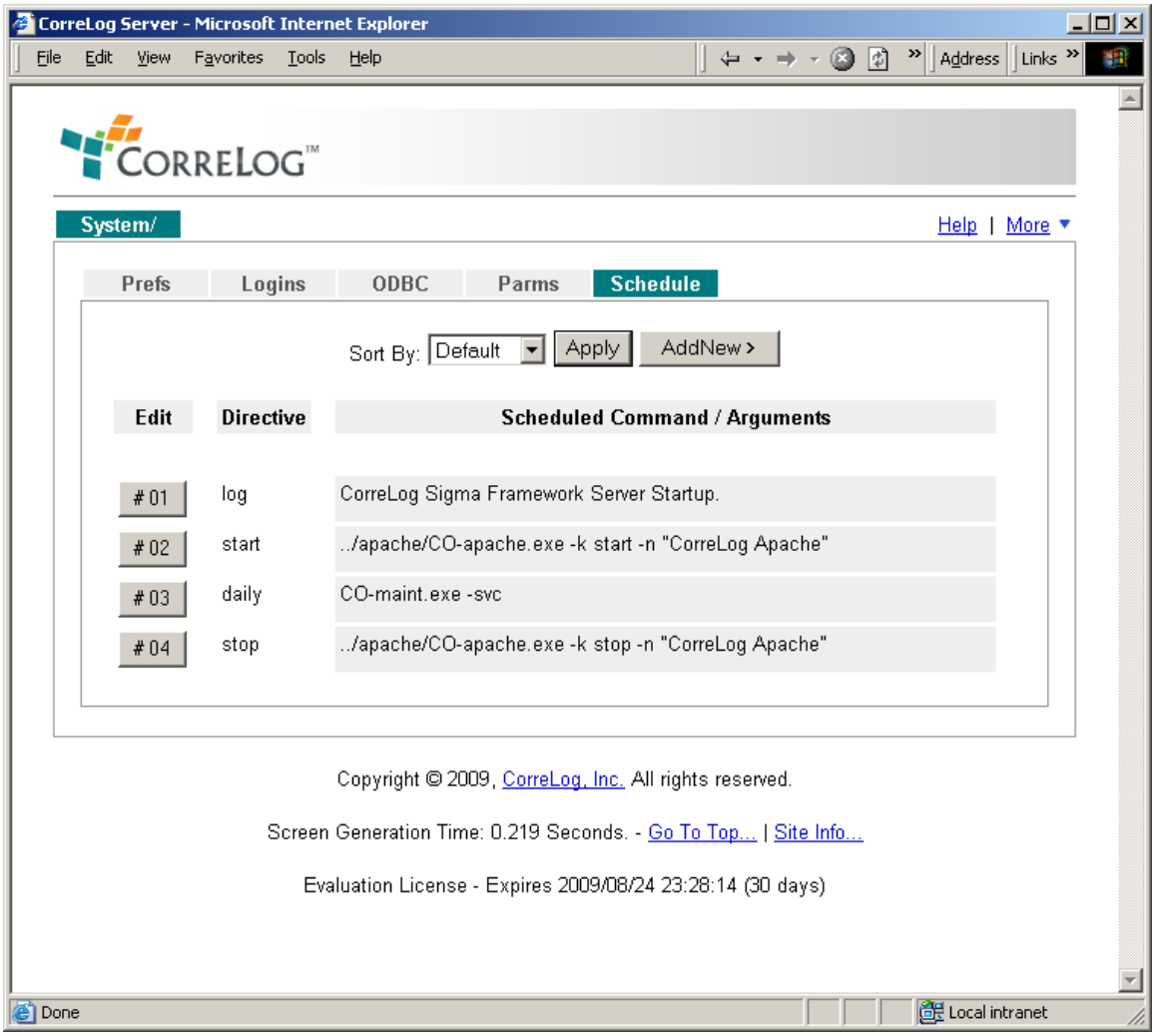
- **Default User Access.** This is the access for any user that is defined by the HTTP server, but not by the Sigma framework. This permits the administrator to implement IIS Windows Directory authentication, and disable any user who is not authorized to use the Sigma system. Or, the administrator can set the default user to “Guest”, and permit any user (registered with IIS or Apache) to access the system with a Guest access. In this situation, any user login that is specifically “Disabled” will still be denied access.

- **System Email Contact.** This is the system contact for the Sigma administrator or work group. The value can be used in framework screens, but is not required or used by the core Sigma Framework system.

Note that the above parameters affect the performance of the entire screen, and apply to all users of the Sigma Framework system.

The Sigma “Schedule” Screen

The Sigma Framework Schedule screen allows an administrator to configure when background processes start and stop, and allows an administrator to configure other background processes to run at specific intervals (possibly to support other Framework screens or functions.) This screen is accessed after login by clicking on the “System” tab, and then clicking the “Schedule” tab (as needed.) The “Schedule” screen is available ONLY to users who have been given “admin” rights to the system. Otherwise, attempting to access this screen will result in a “Permission Denied” screen.



The “Schedule” screen is a standard Sigma Framework dialog. To add a new schedule entry to the system, the user clicks on the “AddNew” button. To edit an existing entry, the user clicks the “Edit” button, followed by the “Commit” button on the edit screen. To delete an existing entry, the user clicks the “Edit” button,

followed by the “Delete” button on the edit screen. Selecting an option from the “Sort By” dropdown menu, and then clicking “Apply” can sort screen fields. By default, logins are sorted in logical order of execution.

The Schedule screen consists of a list of scheduling directives and commands or arguments to the system. Each directive has a compatible argument, as described below:

- **Log Directive**. This directive logs a message to the “system/CO-svc.log” file. The message is provided as an argument to the directive. This directive is processed when the Sigma Framework Service starts.

- **Set Directive.** This directive sets a specified environmental variable to a specified value. The variable / value combination is provided as an argument to the directive using ENVVAR=Value format. This directive is processed only on when the Sigma Framework Service starts.

- **Start Directive.** This directive specifies the name of a system command that is executed when the Sigma Framework Service starts. The system command pathname should be relative to the “Sigma\system” directive, or should be a full pathname to a program, or should be a program in the path of the CO-svc.exe program.

- **Hourly Directive.** This directive specifies the name of a system command that is executed on each hour change. The system command pathname should be relative to the “Sigma\system” directive, or should be a full pathname to a program, or should be a program in the path of the CO-svc.exe program.

- **Daily Directive.** This directive specifies the name of a system command that is executed on each day change, at midnight. The system command pathname should be relative to the “Sigma\system” directive, or should be a full pathname to a program, or should be a program in the path of the CO-svc.exe program.

- **Weekly Directive.** This directive specifies the name of a system command that is executed on each week change from Saturday to Sunday, at midnight. The directive should be relative to the “Sigma\system” directive, or should be a full pathname to a program, or should be a program in the path of the CO-svc.exe program.

- **Monthly Directive.** This directive specifies the name of a system command that is executed on each month change. The system command pathname should be relative to the “Sigma\system” directive, or should be a full pathname to a program, or should be a program in the path of the CO-svc.exe program.

- **Stop Directive.** This directive specifies the name of a system command that is executed when the Sigma Framework Service terminates. The system command pathname should be relative to the “Sigma\system” directive, or should be a full pathname to a program, or should be a program in the path of the CO-svc.exe program.

Each command path may be specified by forward or backward backslashes, and each command can contain zero or multiple command arguments. Commands that are executed at a particular time interval are executed at the same time, as background processes. Hence, programs that may interfere with each other should have some mechanism (such as lock files) to prevent deadlock or contention.

By default, the Sigma scheduler contains those commands needed to execute and shutdown the Apache HTTP server. These commands can be removed from the scheduler if the administrator does not wish to execute Apache (such as if IIS is being used.)

The “hourly”, “daily”, “weekly”, “month”, and “stop” directives may be added to the scheduler without stopping and restarting the system. (The “start”, “log”, and “set” directives, which execute only on Sigma Framework startup, are executed the next time the Sigma service starts.)

Note that these directives are contained in the “CorreLog/config/sched.cnf” file, which is read by the CorreLog/system/CO-svc.exe program. The user can edit the “sched.cnf” file directly as a method of changing these values.

Finally, note that the scheduler function has important application in the Framework, such as in the periodic refreshing and building of screens and data on the system, or implementing cleanup functions. The Framework scheduler is similar in operation to the Windows scheduler, except programs run only when the Framework service has been started.

Section Summary And Additional Notes

1. Sigma comes with some predefined screens, which permit users to modify system logins, modify parameters, and schedule tasks. These are a standard part of all Sigma Framework implementations.

2. The Login screen permits the user to create username and passwords, and assign access rights to users. The passwords are copied to the Apache HTTP server password file. Passwords are encrypted on the disk.

3. If the administrator elects to use IIS or some other web server, the Sigma username still needs to be recorded in the “Login” database to assign

proper access to applications. (The password field may be ignored if the administrator is using IIS authentication.)

4. The Parameters screen affects all users, and allows system colors, CGI timeout, and other parameters to be viewed and modified by the administrator.

5. The “Default Access” parameter can be used to limit access as follows: If a user is not recorded in the Sigma application, that user is assigned a “Default Access” of either “admin”, “user”, “guest”, or “disabled”, where the default setting is “disabled”.

6. The Scheduler program allows the user to schedule periodic tasks, which can be used to perform system maintenance tasks, or can be used to run programs that automatically generate HTML files and screens.

7. All of the screens in this section are accessible ONLY to users given “admin” access to the Sigma interpreter.

This section concludes the general overview of the Sigma Framework from a user’s perspective. The next two sections discuss how to create simple and more complex applications that run in the framework, followed by detailed information on the Sigma Framework C / C++ language API.

Section 4: Simple Applications

This section begins the discussion of actually using the Sigma Framework as a method of presenting data, creating applications, and serving as a methodology and language independent platform for program development. The development of Sigma Applications is presented in the next two sections. First, simple applications will be discussed, and then more sophisticated applications and forms in the sections that follow.

The fundamental purpose of Sigma is to provide a consistent set of rules, procedures, and protocols for rapid development of usable software. As a framework product, Sigma does not enforce any specific programming methodology, or require a specific language to be used. The program's architecture is intended to make development of a web-based framework "simple" for both the developer and the end user.

To that end, Sigma permits the simplest of applications (such as batch files and normal web pages) as well as complex interactive applications to coexist, in modular form, in the framework system. This first section discusses simple, perhaps trivial, application development.

Navigation Tabs

The “Tabbed Navigation” feature of Sigma, where screens and applications are accessed via a series of nested tabbed folders, is one of the simplest and most powerful aspects of Sigma, and is explained as follows:

When Sigma starts, it looks in the “Sigma\sigma-web” directory, and lists all the objects in this directory that follow a naming convention of “nnn_Name.sfx”, where “nnn” is a three digit integer number (with possibly leading zeros), “Name” is the name of the tab, and “.sfx” is the file suffix, defined below.

Any object in the “sigma-web” folder that follows the above naming convention is represented by a (possibly nested) tab, and is accessible to the browser. Any object in the sigma-web directory that does not follow this naming convention is silently skipped.

Supported File Suffixes

The action taken on the actual file depends upon the type of suffix. The following (and only the following) suffix values are supported.

- **.txt (Text File)** Any file with a “.txt” file suffix displayed as preformatted text, bracketed by the “<pre> and “</pre>” tags. The text should be regular ASCII text, and

- **.html (HTML file)** Any file with a “.html” or “.htm” suffix is displayed as a regular HTML file, interpreted by the browser like any other HTML file on the system

- **.exe (System Executable)** Any file with a “.exe” suffix is executed, and the standard output of the program is displayed as regular HTML.

- **.bat (Batch File)** Any file with a “.bat” suffix is executed as a Windows batch file, and the standard output of the program is displayed as preformatted text, bracketed by the “<pre> and </pre> tags.

- **.cmd (Link file)** Any file with a “.cmd” suffix is executed as a Windows batch file, and the standard output of the program is displayed as regular HTML. (This is similar to the action of a “.bat” file, except the output of the program is not treated as preformatted text.)

- **.dir (Directory)** Any file with a “.dir” suffix is taken as a directory. The first file in the directory (listed in the numerical order of the “nnn” prefix) is displayed, which can be another directory or any of the file extensions above.

Users can copy text and HTML files into the “sigma-web” directory, or some new subdirectory within that directory. Those files will immediately appear as tabs. The number of tabs at any level is limited to MAX_LIST_ENTRIES (i.e. 1000 items), although in practical terms, it is typical to have only a few tabs, broken down into folders, for easy navigation through the system. (Otherwise, if there are too many tabs, a horizontal scroll bar will open up in the browser display.)

The system is language independent. In particular a “.cmd” file can immediately launch any other scripting language, including PHP, Ruby, Java, or other application that writes to standard output. This opens the door to support for any programming and scripting language.

The Web.exe CGI Program

Files in the “sigma-web” directory are not accessible directly from the browser. (By default the “sigma-web” folder is not one of the virtual folders of the HTTP server, and it is probably a security concern to make that directory visible in most situations) The processing to create the navigation tabs takes place in the “web.exe” program, launched as a CGI executable by the HTTP server.

The “web.exe” program accepts as an argument the pathname to one of the above objects, starting with the “./sigma-web” directory. For example, to view the “100_Welcome.html” screen, the following URL can be used:

```
http://hostname/s-cgi/web.exe?100_Welcome.html
```

Note that the pathname to the object is revealed (although the location of the “sigma-web” directory is hidden, which is good.) For a better effect, the URL following the “?” character is generally encrypted and encoded, beginning with “//” characters. (This is described in more detail within subsequent sections.)

If the “Web.exe” program is executed with no arguments, it searches the “Sigma\sigma-web” directory for the first file in the directory, possibly recursively descending into any other directories to find that first file. That file is displayed, along with various navigation tabs at the same level as the file.

If the user is launching a program, rather than a file, the URL can contain arguments to the program, separated by ampersands. For example, the URL to launch a program called “120_Hello.bat” the following URL can be used:

```
/s-cgi/web.exe?120_Hello.bat&Arg1&Arg2
```

The above arguments are passed to the %1 and %2 variables of the “120_Hello.bat” file, where they can be processed. Note that the arguments must be simple, or must contain HTML character codes for special characters

(including the HTML “+” character for any spaces.) Also note that the program receives the arguments as double quoted strings (which may cause difficulties in Batch files, but not necessarily in other programming languages.)

Encoding And Encrypting URL Arguments

Because of the difficulties in passing arguments as URLs (which are highly constrained by the HTTP standard to contain only specific characters and whose character symbols are fairly overloaded) it makes more sense to encode all values as a URL “payload”, and pass this payload as part of the URL. This is what users normally see as part of the Sigma URL, i.e. a long string of hexadecimal numbers rather than any path components.

Encoding the URL “payload” provides extra security by obscuring the actual pathname components of the system. It also provides the ability to pass any character regardless of its (possibly special) meaning. For example, Chinese characters can be passed as part of the URL without need for special language support. Binary data (including compressed data) can also be passed using this method.

The Sigma framework encodes the URL payload as a matter of course, rather than relying on the previous methods discussed. (The previous discussion about passing the URL payload is mainly useful to illustrate the operation of the web.exe program, and is usually not practical to actually implement.)

Sigma provides various tools to encrypt URL arguments, including the “system/sigcmd.exe” framework component, discussed in later sections. The user can easily encode any single line of data. The encoding uses a proprietary block rotating, time sensitive, non-repeating cipher, which is highly secure.

The programming techniques to make use of this encoding are straightforward. The program developer either passes at URL argument to a function call or executes the “system/sigcmd.exe” command line utility to encode and encrypt the URL data before it is written to standard output.

Decoding And Decrypting Arguments

No facility is provided, or is required to decode URL arguments. This is because the “Web.exe” program decodes the encoded strings to navigate through the directory structure, and all information is presented to programs in already decoded and deciphered form.

This simplifies the URL encryption, and increases security, by eliminating the need (and possibility) of deciphering the encrypted URL data by application programmers. The decoding and decryption is provided as a built in service of the Web.exe program.

Installing Text Files And Batch Files

If a file with a ". txt" suffix is installed in the "sigma-web" directory, all the text in the file is completely preformatted. Any special HTML characters (such as the "<" and ">" characters) are automatically quoted so that no HTML can be displayed. This is an action similar to that of the standard HTTP server upon encountering a ".txt" file on the system. If a programmer wishes to include any HTML in the file for interpretation by the browser, some other file extension (described below) should be used.

If a file contains a ".bat" suffix is installed in the "sigma-web" directory, the file is executed and the output is bracketed with "<pre> and </pre> characters. Note that this is slightly different from a ".txt" file, in that selected HTML can be coded into the file (such as to change the color of text, as well as perform other functions.) Specific information regarding the "<pre>" tag, and actually preformatted text is available from the HTML standard.

To launch a program that generates HTML, a developer can execute a ".cmd" file, which operates identically to a ".bat" file, but bypasses any generation of "<pre>" HTML tags. This affords an easy and logical way to launch PHP, Ruby, Perl, and other scripts. The working directory for the scripts execution will be the "s-cgi" directory. Care should be given to the setting of the PATHEXT environmental variable to make sure that the proper interpreters are launched.

Execution Of Programs

Any ".exe" file, copied into the "sigma-web" directory with the proper naming conventions, can be executed. If the program generates standard output, then this output is displayed by the browser, and can be HTML. If preformatted text is desired, the ".exe" should be wrapped in a ".bat" file.

To execute scripts, use a ".cmd" file wrapper, possibly launching the script interpreter (such as to generate Perl or PHP or Ruby output.) This provides a very simple way of integrating executable console programs and batch files into Sigma.

Note that if the program generates error output, this may appear asynchronously in the display, before the standard output. The particular order of output may become complicated, and require a ".bat" or ".cmd" wrapper for the command in order to control the output stream.

Note that any program executed by Sigma must be transitory in nature. The CGI timeout value, discussed in a previous section, limits the execution of programs to five minutes, normally only 30 seconds. Good development practices dictate that CGI scripts and programs execute as quickly as possible.

Installing HTML Files

The Sigma interpreter permits any HTML file to be accessed. The file can contain either a “.html” or a “.htm” extension. The raw HTML data (with the exception of "macro replacement" characters, noted below) is displayed without processing. These can be any standard HTML files, including HTML directives, JavaScript, or ASP pages. The user simply places the HTML file in the "sigma-web" directory, with a names such as "001_Myfile.html". All the text contained in the file is relayed to the web browser without any modification, with the exception of macros, as follows:

As a special facility, the Sigma interpreter incorporates a “macro replacement” function, which will substitute certain words in the HTML file (which are delineated with “@@” characters) with dynamic values, read from a configuration file, or read from an environmental variable. This allows users to substitute sections of text with other text, contained in a separate file.

The Sigma Macro facility greatly expands the ability to make dynamic web pages. The user can contain certain information in a separate file, or in environmental variables. The user can also execute programs using the macro facility, when the web pages are displayed, and the output of these programs is automatically incorporated into the web page.

Using Sigma Macros In HTML files

The "Macro" function uses a straightforward substitution, where any text string within the HTML document, delimited by @@ characters, is replaced by another value. This value can be an environmental variable value, or can be a text string contained in a configuration file.

The configuration file can be dynamically generated, or can be static to simplify maintenance of the program. The specific operation of this facility is described below:

1. If a “.cnf” file, with the same name as the specified file, resides in the directory with an HTML file, that file is opened and read by the “web.exe” program. The “.cnf” file must contain a list of “tokens” followed by text and a new line.

2. The first token in the configuration file must consist of a word with letters, numbers, underscores, dashes, but no spaces. The first non-space character following the token is taken as the start of the value, until the end of the line. Examples of valid configuration files, which implement this format, can be found in the “CorreLog/config” directory of the installation

3. As the HTML file is scanned, any occurrence of “@@token@@” characters in the HTML file are immediately replaced by the value of the token (if any) read from the configuration file. The @@token@@ is eliminated, and immediately replaced with the macro value before the HTML is displayed.

For example, a web page named "sigma-web/001_Myfile.html" may have an accompanying file "sigma-web/001_Myfile.cnf", which contains a list of macro names and values. When the "Myfile" tab is clicked, the web page is displayed, and any macro definitions in web page (delimited by "@@" characters) will be substituted with the values contained in "001_Myfile.cnf".

The above facility permits a general sort of simple dynamic HTML to be implemented, where specific HTML documents can contain variable fields. The “.cnf” files can be maintained as part of the framework, and can be dynamically generated (such as using the Sigma Scheduler facility, mentioned in a previous section.) This provides a very simple and straightforward way of making dynamic web pages without any strenuous programming required.

Note that these macro files are currently limited to only one line of content, without any ability to span multiple lines. (Each line of the macro file is necessarily delimited by a new line character, hence cannot span multiple lines of text.) Each macro name is limited to 50 characters, and the replacement text is limited to 500 characters.

Finally, note that the actual content of the "001_Myfile.cnf" will not otherwise be visible to the user (since the ".cnf" extension is not supported by Sigma, hence no tab will be created for this file.)

Sigma Macros And Environmental Variables

In addition to permitting macros to be stored in an associated configuration file, Sigma also permits substitution of values using environmental variables. When parsing and displaying an HTML file, if a @@token@@ type symbol is found, and the value of “token” is not contained in the configuration file, then an environmental variable with the value of “token” is used (if such an environmental variable exists.) This provides a convenient way of processing HTML files from within scripts, especially “.bat” and “.cmd” files (but also from Perl, Ruby, and Java script programs.

For example, the value of @@REMOTE_USER@@ will always be substituted in an HTML file for the name of the remote user, logged into Sigma. (This is because the REMOTE_USER environmental variable is always set via the HTTP server as part of the execution environment.)

If a @@token@@ symbol is found in an HTML file, and the “token” is not found in either the configuration file, or as an environmental value, the entire token is simply deleted from the file and is not displayed as part of the HTML output.

Sigma Macros And External Program Execution

The Sigma Macro capability allows external programs to be executed when an HTML page is accessed. The standard output of these programs incorporated into the HTML output, permitting the user to create dynamic pages using scripts or ".exe" files.

The special "@@!" character (i.e. the standard macro delimiter, followed by an exclamation mark) is handled as a special case by Sigma. Rather than replacing the macro with a value contained in an environmental variable, or in a configuration file, the Sigma program takes the text following the exclamation mark, regards it as an external program and arguments, and passes this to the system for execution. Any resulting output of the program replaces the macro name.

For example, incorporating the text string "@@!netstat –s@@" into an HTML document causes the system "netstat –s" program to be executed, and the output of this command to replace the macro name. The macro can be bracketed with "<pre>" and "</pre>" tags to correctly display the preformatted text output of the command.

This technique can also be used to insert external text and HTML files into an HTML document, using a command such as "@@!type .c:\myfile.txt@@", which uses the standard MS "type" command to display the "C:\myfile.txt" file to standard output, thereby incorporating it into the HTML document.

The working directory for Sigma macros is the "cgi" directory of the Sigma system. Any pathnames should be absolute, or relative to the "cgi" directory. The Sigma "system" directory is always searched first for any external command.

Note that, when using the special "@@!" type macro, the executable command should not block, cannot require any interactive user input, and must execute quickly to allow acceptable web page response time. Further note that only the standard output of the command is incorporated into the web page, and any standard error output is discarded.

Finally, note that the "@@!" type macro has special implication with the "RunSQL.exe" facility, and is a very simple way of displaying the output of SQL queries on a database. (See the Appendix on the Sigma SQL interface for specific information.)

CGI Headers And META HTML Tags

The “Web.exe” program takes care of the generation of all CGI headers. Users can insert JavaScript, or change the style sheet, using the following special file.

```
Sigma\s-html\s-header.html
```

This file, if it exists, is displayed to generate the HTML header for the system, which can incorporate meta tags, JavaScript, script includes, or other special markup that may be relied upon by Framework applications. A single instance of the file exists, and affects all framework programs. This provides a simple way to make global changes to the system, including specifying different style sheets. However, because this single header file exists for all applications, it should be modified ONLY with caution, to prevent affecting other framework programs that may be using the file..

Note that, in addition to the s-header.html file, there is an accompanying file that is executed immediately before the “Web.exe” program terminates. This file is as follows:

```
Sigma\s-html\s-footer.html
```

The above file complements the s-header.html file, but is less useful for programmers. It is mainly useful for creating a footer that displays program copyrights, and possible generates links to access special screens.

If either the s-header.html or s-footer.html file is missing from the system, the Web.exe program creates a default generic header automatically.

Section Summary And Additional Notes

1. Developers can create subdirectories within the “sigma-web” directory, following a strict naming convention, to create navigation tabs.

2. The “sigma-web” directory is not accessible to the browser, except for the Sigma Framework “Web.exe” program, residing in the “s-cgi” directory.

3. The Sigma Framework supports “.txt”, and “.bat” files, which are used to display preformatted text.

4. The Sigma Framework supports “.exe”, and “.cmd” files, which are used to display HTML screens.

5. The Sigma Framework supports “.html” and “.htm” files, displaying these files, possibly with macro substitution of keywords and content, based upon an external “.cnf” file.

6. The Sigma Framework macro editor permits macro substitution of tags, including the execution of external program. Macro tags are delimited by "@@" characters, and the special "@@!" delimiter permits execution of external ".exe" programs and batch files.

7. URLs can be encrypted using the “system/sigcmd.exe” utility, or using the Sigma programmer API, discussed in later sections.

8. No capability (or need) exists to decrypt the encrypted URL payloads, which are decoded and deciphered by the “web.exe” program prior to executing batch files or programs.

9. Developers can modify the “s-header.html” and “s-footer.html” files to add HTML "meta" tags, specify JavaScript header items, and specify global program style sheets.

The next section describes more advanced application development, including examples of how to use the Sigma Framework “sigcmd.exe” utility to create interactive forms. The next section also provides an introduction to the Sigma Framework API, discussed in detail in later sections of this manual.

Section 5: Advanced Applications

This section continues the discussion of creating Sigma Framework applications. More advanced, interactive applications are discussed, including how to create interactive forms and pass CGI arguments to programs via “Post” commands.

Sigma provides an easy framework for creating these types of applications, and provides facilities and services to pass CGI arguments and HTTP “Post” data to subprograms launched by the “Web.exe” interface. Using these facilities, programmers can create stand-alone applications that work outside of Sigma, or integrated applications that leverage the “sigma.lib” functions of the Framework programmer SDK.

This section provides a discussion of the advanced features, and an introduction to the Sigma.lib programmer API. It contains additional information that will be of interest to script authors and intermediate programmers.

Additional information on the Sigma API, including the “sigcmd.exe” program and the "runsql.exe" program, is available in the Appendices to this document, including a description of all configuration files on the system.

Acquiring CGI Values

The Apache HTTP server essentially requires the user to collect CGI arguments prior to generating any output. This means that the “web.exe” program must gather the CGI arguments BEFORE it launches a subprogram (or script), and must provide a mechanism for passing these arguments to the executed program. Failure to comply with this rule will cause the HTTP server (and web browser) to mysteriously “jam” and hang when the number of CGI arguments reaches the buffer space limit of the web browser, preventing any further output of the HTML server.

The “Sigma.lib” provides a simple set of calls to collect these CGI post values, and allow these values to be fetched by name. These CGI post values are also available to scripts and batch files by accessing the S_POST environmental variables, which contains the name of a temporary file that holds the CGI arguments posted to the HTTP server.

S_POST Environmental Variable

The S_POST environmental variables point to a temporary file with the name “temp/cgi.nnnn” where “nnnn” is the PID for the executing “Web.exe” program. (This name is generally not important, but provided here for completeness.) This temporary file exists only as the application program executes, and is automatically cleaned up by the Sigma Framework when a screen is displayed.

The S_POST environmental variable uses forward slashes to represent the temporary filename. An alternate variable, S_WIN32_POST, points to the same file, but uses backwards slashes. (This makes scripting of “.bat” and “.cmd” files easier.)

The actual CGI temporary file consists of a list of form input names followed by their values, where each value is delimited as the first non-space following the input name. This is the standard configuration file for Sigma, introduced earlier. For examples of this type of configuration file, inspect the “CorreLog/config” directory.

The programmer can access the S_POST (or S_WIN32_POST) environmental variable, open the file, and rapidly parse arguments. This technique is suitable for use with a variety of scripting languages, including Windows Batch files.

For C and C++ programmers, the sigma.lib function “get_cgi_args()”, discussed in the appendix of this document, will be more convenient, and includes the ability to fetch any posted value directly by input name, as well as copy configuration items, check for pattern matches in configuration items, and various other utility functions.

Implementing A Batch File Example

By way of example, consider a simple HTML input form constructed from a ".bat" file. This program generates an HTML form, and then displays the values filled in by the user when the submit button is clicked.

In addition to illustrating the mechanism for posting forms and getting values, this example further illustrates an important feature of the Sigma Framework, that is, the ability to directly program with such rudimentary tools as a Windows CMD.exe program.

Below is the example command file, residing in the "sigma-web/100_Form.cmd" directory. This program, by being located in the "sigma-web" directory and following the prescribed naming conventions, creates a tab at the top of the display called "Form". The ".bat" file is quite simple, and contains the following few lines:

```
@Echo off
REM: Filename: sigma-web\100_Form.cmd
Type \data\form.txt
Type %S_WIN32_POST%
```

The first line uses the "type" command to display the HTML for the form. That HTML resides in the file \data\form.txt, which is a small form with three input fields and a submit button as follows:

```
<!-- Filename: \data\form.txt -->

<form method=post
action=/s-cgi/web.exe?100_Form.cmd>

Arg1: <input name=arg1 value=""><br>
Arg2: <input name=arg2 value=""><br>
Arg3: <input name=arg3 value=""><br>
Submit data: <input type=submit>

</form>
```

When the user access the "Form" tab at the top of the display, a form with three input fields and a submit button is depicted. (The HTML is generated by the CMD

“type”, which is a standard Win32 command for displaying data to standard output.)

When the user clicks on the “Submit” button, the data is delivered directly to the S_WIN32_POST file as a series of input name and value pairs. Then, any contents of the %S_WIN32_POST% file are displayed to standard output. When the user clicks the submit button, the user sees the values that were posted to the form.

Although the above program is very simple, one can see how it could be made more powerful by replacing the second “Type” command with an external “.exe” program, which might input the data to a database, or append the data to a particular file, or generate specific HTML based upon the user input values. An even more obvious enhancement: the user could add a second “type” command that appends any posted data to a file, such as by adding the final command to the batch file as:

```
type %S_WIN32_POST% >> logfile.txt
```

The above command would post any arguments to the “logfile.txt” file, possibly consisting of e-mail addresses or contact information from a website.

Note that there is no need to delete or unlink the file %S_WIN32_POST% – this will be taken care of by the “Web.exe” program after the “100_Form.cmd” program is finished.

S_ACCESS Environmental Variable

Whenever a logged in user access a Sigma Framework screen, the value of the REMOTE_USER environmental variable is available as a standard service of the HTTP server. Additionally, as a feature of the Sigma Framework, the particular access rights for that user are fetched from the “CorreLog/config/pass.cnf” file, and assigned to the value of S_ACCESS. This environmental variable will have the values of “admin”, “user”, or “guest”, depending upon the access that has been assigned to the user via the Login Editor screen, discussed in a previous section.

The S_ACCESS variable can be easily used to permit access to certain screens, based upon the permissions of the user. For example, in a Windows Batch file, the following lines (at the top of the batch file) will limit execution of the file to those who are Sigma Administrators.

```
@Echo off
IF “%S_ACCESS%” ==“admin” goto CONTINUE
type ..\s-html\error-access.html
exit
:CONTINUE
Echo You are authorized to continue execution.
```

As depicted above, the batch file uses the value of the S_ACCESS environmental variable to branch the program to the CONTINUE statement, only if the currently logged in user has been assigned a role of “admin” on the Sigma Framework login screen. Otherwise, the “s-html/error-access.html” page is displayed (which is the standard Sigma page displayed when permission to a screen is denied.)

In the C / C++ SDK, two principle functions are provided, as discussed in the appendices. These functions are usually found at the start of any program, and limit the access of who can execute these programs. These are the “require_admin()” function and the “deny_guest()” function. Refer to the appendices for specific information on usage of these routines.

Encoding The HTTP POST Action URL

In the previous examples, the URL payloads have not been encoded. The data appears in clear text as part of the Web browser “Address” line. This may or may not be a security issue. However, it is very simple to encode the URL payload, using the “sigma.lib” functions, and “Sigcmd.exe” program.

This encoding, as discussed previously, provides several advantages, and is well worth the minimal effort required by the developer if the application is to have a substantial degree of sophistication and utility.

First, by encoding and encrypting the URL, the operation of the Sigma Framework is obscured to end users, which will enhance the general security of the program, and promotes the basic concepts of “data privacy.”

Secondly, encoding of the arguments prevents special HTML characters (such as spaces, ampersands, semicolons, etc.) from interfering with the normal operation of the user’s web browser. This permits data such as ISO characters (for example Chinese characters) to be passed to receiving programs.

In particular, if the programmer encrypts the URL payload, data is handled much easier. The programmer does not need to worry about special quoting of

characters, or formatting the URL payload solely to permit the URL to be passed from the web browser into the program.

Consider yet another simple batch file example, which demonstrates how to encode the URL, so its value is hidden from the browser user. The program below queries an e-mail address, and a comment, from a user, and appends this information to the “\data\comments.txt” file on the system.

```
@Echo off

REM: Create the form tag.
sigcmd.exe SG_form %0%

echo Enter e-mail address: ^<br^>
echo ^<input name=email value="" ^>
echo ^<p^>

echo Enter comment: ^<br^>
echo ^<textarea name=comment ^>
echo ^</textarea^>
echo ^<p^>

echo ^<input type=submit ^>

echo ^</form^>

REM: Append any posted data to this file.
type %S_WIN32_POST% >> \data\comments.txt
```

Several things can be noted from the above example. First, observe that because it is a batch file, and the “<” and “>” characters have special meaning, these characters must be escaped using the “^” character. (This, of course, makes the Windows batch file programming less than ideal for scripting screens, but is good for illustrative purposes.)

The “sigcmd.exe” program is documented in the next section, and contains various utilities that correspond to the “sigma.lib” programmer’s library. This utility provides various useful functions. The “sigcmd.exe SG_form” command, issued at the top of the program, creates a <form> tag, where the action of the tag is given as the first argument. This action is encrypted. In this case, the action is to execute the above batch file (as given by the “%0%” argument, which is the name of the currently executing program.

When the use clicks the submit button, the program is re-executed by the HTTP server. Any submitted data is placed in the \data\comments.txt file of the system.

Note that, because the argument to the SG_form command is “%0%”, the above batch file can be placed in any directory within the “sigma-web” folder, with the following two provisions:

1. The “sigcmd.exe” program must appear in the path of the HTTP server. The utility normally resides in the “sigma\system” directory, and the PATH variable should either be modified to include this directory, or the “sigcmd.exe” program should be copied to some location already in the HTTP server path (such as the %SystemRoot% directory.

2. The batch file should follow naming conventions, including a “.cmd” suffix. (Note that a “.bat” suffix will not work, since these types of batch files display preformatted text and not raw HTML.)

The fact that the “%0%” argument is passed to the SG_form command makes the program completely relocatable, so long as the program resides within the “sigma-web” directory. This means that, if the file is named “900_Comments.cmd” (for example) the file can be copied into the top-level “sigma-web” or lower subdirectories of that folder, and still work without modifications.

Encoding Arbitrary URLs And Arguments

In the previous example, the Action URL of a form statement was encoded. It is also possible (and sometimes necessary) to encode normal URLs to access Sigma screens, including passing data as part of the URL. This is easily accomplished using the “sigma.lib” functions and “sigcmd.exe” program, using the “SG_link” function.

The SG_link function is used to encode any URL that is within the Sigma framework. It is not useful for accessing files outside of Sigma (because the URL “payload” is encoded and will not be decipherable to other applications.) It is an easy way to construct hypertext anchors in the form “<a href=url>”, where the “url” value may contain additional command arguments passed to the target program.

To illustrate the usage of the SG_link function, consider the following batch file, which resides in the “sigma-web\010_DisplayArg.bat” file.

This file creates a “DisplayArg” tab, which when clicked by a user, performs no substantial action. However, if a URL accesses the program, and that URL contains one to four arguments, those arguments are displayed to the user.

This can be used to illustrate how a URL can not only execute a program, but also pass data to that program similar to the way a form post operates. For

example, a URL may contain data that is read as the command arguments of the executed program. (In this case the executed program simply displays the arguments, but more realistic uses of program command arguments are obviously known.)

```
@Echo off
REM: Filename: sigma-web\010_DisplayArg.bat
REM: Display the first four command arguments.

Echo Program Pathname: %0%

Echo Arg1: %1%
Echo Arg2: %2%
Echo Arg3: %3%
Echo Arg4: %4%
```

The above batch file simply displays the first four arguments passed to the program, and is useful to illustrate the operation of passing arguments (if not possessing any other use.)

Now consider that the user wants to execute the 010_DisplayArg.bat file from another screen, passing that target various arguments. The program to accomplish this is provided as “005_SetArg.cmd”, as shown below:

```
@Echo off
REM: Filename: sigma-web\005_SetArg.cmd
REM: Pass arguments to the 010_DisplayArg.bat file.
REM: An encrypted URL passes these arguments.

Set URL= 010_DisplayArg.bat
Set URL=%URL%;My first argument
Set URL=%URL%;My second argument
Set URL=%URL%;My third argument
Set URL=%URL%;My last argument

sigcmd.exe SG_link “%URL%”
Click to access DisplayArg screen.
Echo ^</a^>
```

The “005_SetArg.cmd”, program, above, creates a link. The link consists of various arguments (in addition to the URL to a target program.) When the link is clicked, the target “010_DisplayArg.bat” program is launched, and that target

batch file receives the various arguments defined above. The arguments are received as normal and unencrypted arguments %0%, %1%, etc.

Several things should be noted that are somewhat critical: First, the various arguments, appended to the URL value, are delimited by semicolons, and not by '&' characters. This is because the "&" character is reserved for passing data WITHOUT encoding, as discussed in a previous chapter. This may be an easy programming mistake to make.

Also, note that the SG_link program expects a single parameter, hence the URL is enclosed in double quote marks. (This would not be necessary, except that the various arguments contain spaces.)

Finally, note that the opening anchor <a> is created by the SG_link function, but the user needs to supply the closing anchor. Failure to do so will result in all subsequent text being incorporated as part of the link (a common programming mistake.)

Section Summary And Additional Notes

1. The Sigma framework contains specific elements needed to receive posted arguments, required to make forms and more advanced screen systems.

2. When a form is posted, the input form names and values appear in the temporary file given by the environmental variable S_POST. This file contains a list of input names and values that are easily parsed and read by any program.

3. The S_WIN32_POST environmental variable is identical to the S_POST variable, except that the pathname to the temporary file is represented as back slashes rather than forward slashes. Both of these environmental variables point to the same file.

4. The S_ACCESS environmental variable contains the currently defined access of the logged in Sigma user, and can be used to limit access and execution of screens within the Sigma Framework.

5. The "sigcmd.exe" program contains various utilities, documented in detail in later sections, that make the encoding and encryption of URL data easy. The executable corresponds to C++ language function calls available to programmers in the "sigma.lib" library, documented elsewhere.

6. The sigcmd.exe SG_form command (and SG_form() function call) creates a form HTML tag, where the form action payload is encrypted. (See next section for more information.)

7. The sigcmd.exe SG_link command (and SG_link() function call creates a hyperlink anchor, possibly containing additional arguments passed to the target, where the URL payload is encrypted. (See next section for more information.)

8. When using the SG_link function, semicolons, and not ampersands MUST separate the command arguments passed to the target program. Each argument is appended to the target, separated by a ";" character.

9. It is the responsibility of the programmer to close the form and anchor tags generated by the SG_form and SG_link calls.

In this section, various techniques have been illustrated using Windows batch files, principally because this is a very understandable demonstration of how the Framework operates.

Although developers are free to make liberal use of batch files, in practice most systems will make use of other, more robust programming languages such as Perl, PHP, Ruby, Java, or .NET. In particular, the Sigma framework contains a comprehensive C / C++ programming API, which is reflected in the "sigcmd.exe" utility, and which is documented in detail within the next section.

Section 6: Framework API

In the previous section, the use of simple scripting languages (in particular, Windows Batch files) was discussed. This section begins the discussion of how to use the Sigma Framework Application Program Interface (API) calls to construct programs in C / C++, other compiled languages.

- **Configuration Data Functions.** The Sigma Framework contains various routines that permit the reading, writing, setting, and fetching of configuration data from files. This configuration data can be used as a global data store.

- **HTML Generation Functions.** The Sigma Framework contains various routines that assist in the generation of HTML, including the ability to create encrypted URLs, Select Menus, and other markup code.

- **Miscellaneous Utility Functions.** The Sigma Framework contains various routines that provide general utility in processing string text, acquiring directory lists, manipulating time values, and performing system house keeping.

This section will mainly be useful to application developers, wishing to add new screens to the Sigma Framework. It is written as a brief command reference, where each function is identified in terms of its inputs, outputs, and return values.

The “Sigma.dll” And “Sigma.lib” Program Files

The Sigma Framework provides the “sigma.dll” program file within the “system” directory of the installation. This directory is ALWAYS included as the first directory searched by the “web.exe” program, so the DLL is always loaded and accessible without need to qualify the program paths.

The “sigma.lib” link library, and include files needed to use this library, are available on selected systems in the “sigma\dev” directory, along with additional resource information to developers.

Note that various versions of these libraries exist, depending upon whether the developer requires single threaded, or multi-threaded support for their application programming.

Configuration Data Functions

The Sigma Framework provides various functions needed to manipulate configuration data. These functions read a configuration file, and create a linked list of name / value pairs. The values can be further assigned, fetched, and saved to external files.

struct list_s *SG_allocate_cf ()

This function allocates a handle to a configuration data item. The function must be called to create the handle, which is subsequently used in all the other functions described in this section. The function returns a pointer to the allocated data.

int SG_read_cf (struct list_s *handle, char *filename)

This function reads the configuration data from filename, and assigns the data values to the list specified by handle (previously allocated by the SG_allocate() function.

int SG_save_cf (struct list_s *handle, char *filename)

This function saves the configuration data specified by “*handle” to the specified filename. This permits a programmer to save configuration data to a file.

char *SG_get_cf (struct list_s *handle, char *itemname)

This function returns a pointer to the text string corresponding to *itemname”, in the specified list *handle. If the itemname is not found, a pointer to a zero length string is returned.

char *SG_get_cf_name_by_smatch(struct_list_s *handle, char *patt)

Similar to the “SG_get_cf()” function, except rather than specifying an item by name, the item is specified as a wildcard. The first matching item

NAME (not value) in the list is returned. This is especially useful for determining button clicks in a CGI program (after acquiring *handle data via the SG_get_cgi_cf() function described below.)

int SG_set_cf (struct list_s *handle, char *itemname, char *itemvalue)

This function sets the specified itemname to the specified itemvalue in the specified list *handle. If the itemname already exists, it is overwritten with the new itemvalue. Otherwise the new itemname is created.

int SG_get_cgi_cf (struct list_s *handle)

This function is called to acquire a list of CGI items and values from the HTTP server. Once acquired, the program can access the individual items using the CG_get_cf() function. This provides a simple way of getting CGI arguments into an application program.

int SG_get_dirlist_cf (struct_list_s *handle, char *dirspec)

Get a directory list, specified by the directory or wildcard *dirspec, into the specified list handle. The value of *dirspec is a pathname to a directory, and can include wildcards such as “*” or “???”. The list is sorted alphabetically.

void SG_free_cf (struct list_s *handle)

Free any data associated with the specified handle. This is added for completeness, and may not necessarily be called in a transient CGI program (where the data is freed when the CGI program exits the system.

HTML Generation Functions

The Sigma Framework provides various functions needed to generate HTML, especially useful for generating markup code containing encoded URLs. These functions generate opening tags and data. The user is required to close the tags later in the program.

int SG_init (int argc, char **argv)

This function reads configuration data from the “./config/sparm.cnf” file, starts the watchdog timer for the CGI timeout, and performs other housekeeping function. Although not required, it is a good idea to start each CGI program with this function call. The argument accepts the command arguments from the main program.

int SG_form (char *action)

Generate the <form method=post action=action> tag, which starts the form. The payload of the specified action is encrypted. This provides a simple method of starting forms. The user is obliged to supply the “</form>” closing tag at the end of the form.

int SG_link (char *url)

Generate the tag, which provides the anchor value for the URL. As discussed elsewhere, the url specifies the target document or program, and can also contain arguments passed to a program, where each argument is delimited by a ";" semicolon. The user is obliged to supply the "" closing tag at the end of the URL hyperlink.

int SG_linkt (char *url, char *target,)

Generate an anchor in the form , which operates in a fashion identical to the SG_link function, but which incorporates the "target" attribute (useful for opening documents in new windows.) The user is obliged to supply the "" closing tag at the end of the URL hyperlink.

int SG_output_html (struct list_s *handle, char *filename)

Output the HTML file at the path specified passing the data through the "macro replacement" function, discussed in a previous section. This provides a method of displaying HTML based upon macro values contained in an external file. Each macro is first checked in the *handle, then in a configuration file with the same basename as the HTML file, then against a currently set environmental variable. The value of *handle can be NULL.

Miscellaneous Utility Functions

The remaining functions in this section are simple utility functions that can be used to facilitate the programming effort. These functions may or may not be required by an application program. In some cases, users may wish to substitute their own commands and functions, depending upon the requirements of the application.

int SG_get_date (char *date)

Get the date and time, in yyyy/mm/dd hh:mm:ss format. This date and time is the standard time element used by the Sigma library, and is used by the SG_diff_date() function described below.

int SG_diff_date (char *date1, char *date2)

Return the number of elapsed seconds between the two dates, where each date is the format described by the SG_get_date() function above. This is useful for determining the elapsed time between any two timestamps.

int SG_run_program (char *syscmd)

This function operates similar to a "system()" function call, but should be used to launch any program within Sigma. In certain situations, the command makes use of a system server process to actually execute the

command (required for proper IIS operation.) Otherwise, the above program operates identically to the standard system() function call.

int SG_smatch (char *patt, char *string)

Return True (1) if the specified pattern exists in the specified string. The pattern can contain the standard “^”, “*”, and “?” wildcards. This is a way of quickly testing whether a specified pattern exists in a specified string.

int SG_get_basename (char *path, char *basename)

Given a pathname, return the basename, which is the name of a file without the leading path or trailing suffix. This provides general utility in processing filenames, in particular configuration file names.

char *SG_get_tab_on_color ()

Return a pointer to a string indicating the color of selected navigation tabs configured on the Sigma Framework “Parms” screen. This function should be used to acquire color values when creating HTML components.

char *SG_get_tab_off_color ()

Return a pointer to a string indicating the color of UNSELECTED navigation tabs configured on the Sigma Framework “Parms” screen. This function should be used to acquire color values when creating HTML components.

char *SG_get_text_on_color ()

Return a pointer to a string indicating the color of selected navigation tab text configured on the Sigma Framework “Parms” screen. This function should be used to acquire color values when creating HTML components.

char *SG_get_text_off_color ()

Return a pointer to a string indicating the color of UNSELECTED navigation tabs text configured on the Sigma Framework “Parms” screen. This function should be used to acquire color values when creating HTML components.

char *SG_get_default_access()

Return a pointer to a string indicating the particular default user name, configured on the Sigma Framework “Parms” screen. This function is provided for completeness.

char *SG_get_email_contact()

Return a pointer to a string indicating the particular System e-mail contact, configured on the Sigma Framework “Parms” screen. This function is provided for completeness.

void SG_require_admin ()

Check the current S_ACCESS value. If it does not indicate that the currently logged in user is an administrator is logged into the system, immediately exit the program and display the “s-html/error-access.html” screen. This function can be placed at the start of a CGI program to ensure only Administrators can access the screen.

void SG_deny_guest ()

Check the current S_ACCESS value. If it indicates that the currently logged in user is a guest, immediately exit the program and display the “s-html/error-access.html” screen. This function can be placed at the start of a CGI program to ensure guest logins cannot access the screen.

The Sigcmd.exe Utility Program

The “system\sigcmd.exe program permits application programs and shell scripts to access particular functions in the library, providing a simple API for scripts, including batch files. This utility accepts as input the name of a function, and the arguments appropriate for that function. Any appropriate return value is written to standard output. If the function name is not found, the utility continues without generating any output, and the return status of the program is set to one (1).

The following syntax elements are available from the Sigcmd.exe program.

Sigcmd.exe SG_form (action)

Generate, to standard output, the <form method=post action=action> tag, which starts the form. The payload of the specified action is encrypted. This provides a simple method of starting forms. The user is obliged to supply the “</form>” closing tag at the end of the form.

Sigcmd.exe SG_link (url)

Generate, to standard output, the tag, which provides the anchor value for the URL. As discussed elsewhere, the url specifies the target document or program, and can also contain arguments passed to a program, where each argument is delimited by a “;” semicolon. The user is obliged to supply the “” closing tag at the end of the URL hyperlink.

Sigcmd.exe SG_linkt (url) (target)

Similar to the above function, but permits the user to specify a target for the anchor. Generate, to standard output, the tag, which provides the anchor value for the URL. As discussed elsewhere, the url specifies the target document or program, and can also contain arguments passed to a program, where each argument is delimited by a “;” semicolon. The user is obliged to supply the “” closing tag at the end of the URL hyperlink.

Sigcmd.exe SG_output_html (htmlfile)

Display the specified HTML file to standard output, performing macro replacement on macro tokens in the form “@@token@@”. Macro tokens are replaced in the file as follows: First check for a file next to the HTML file, but with a “.cnf” suffix. If found, lookup tokens in that file. If no macro replacement is found, check for an environmental variable with the macro name. If neither is found, the macro is removed from the file. (See section on Sigma macro replacement.)

Sigcmd.exe SG_get_date

Display, to standard output, the date and time, in yyyy/mm/dd hh:mm:ss format. This date and time is the standard time element used by the Sigma library, and is used by the “diff_date” function below.

Sigcmd.exe SG_diff_date (date1) (date2)

Display, to standard output, the number of elapsed seconds between the two dates, where each date is the format described by the SG_get_date() function above. This is useful for determining the elapsed time between any two timestamps.

Sigcmd.exe SG_smatch (patt) (string)

Display, to standard output, the text string “True (1)” if the specified pattern exists in the specified string. The pattern can contain the standard “^”, “*”, and “?” wildcards. This is a way of quickly testing whether a specified pattern exists in a specified string. In addition to returning the match status to standard output, the command returns an error level of one if the specified pattern does not match the specified string.

The RunSQL.exe Utility Program

The Sigma system provides a comprehensive interface to SQL databases using the "RunSQL.exe" program, found in the "system" directory of the Sigma root directory. This utility permits users to run arbitrary SQL statements.

Using the Runs program in conjunction with the Sigma "Macro" capability (described in Section 4 of this manual) allows developers to easily create web pages containing data directly acquired from any ODBC database, including Oracle, Microsoft Access, and Microsoft SQL databases.

The RunSQL.exe program requires the user to create a configuration file defining the ODBC interface to operate upon. The configuration file can also contain the query to execute, or the query can be placed on the command line. The actual syntax of the Runs program takes two forms:

Runsql.exe (configfilepath)

This form of the command requires the pathname to a configuration file to be specified. The configuration file contains "dsnname", "username", "password", and possibly other optional directives. In this form of the command, the "sqlstmt" directive MUST be included, and contains a standard SQL query that is executed by the RunSQL program.

Runsql.exe (configfilepath) (sqlstmt)

This form of the command requires the pathname to a configuration file, identical to the above form, but also accepts an SQL query as part of the command line. The query is limited to 500 characters or less. If a query is specified in the configuration file, the command line query is ignored.

Note that the pathname to a configuration file is required in both forms of execution. A detailed discussion of the Sigma SQL interface, including all the RunSQL configuration file directives, as well as application notes, is provided in Appendix B of this manual.

Section Summary And Additional Notes

1. The Sigma Framework provides the Sigma.dll, which resides in the “system” directory of the installation. This directory is always added to the search path of the program by the “web.exe” program, hence this DLL is available to all programs in the framework.

2. On selected systems, the “Sigma.lib” and include files are provided in the “sdk” directory of the installation, which permits C and C++ programmers to access framework functions needed to perform a variety of activities.

3. The Sigma Framework API contains components needed to read and process configuration data, generate HTML, and perform special utility functions for a programmer.

4. The Sigma Framework includes the “Sigcmd.exe” utility, which resides in the “system” directory of the installation. This utility provides functions that are useful to script programs, including Windows batch file scripts. This utility accepts as input the name of an API function, and displays the return value to standard output.

5. The Sigcmd.exe program provides the ability to encode URL components, access to the HTML macro replacement functions, and a variety of other functions that can accommodate simple and more sophisticated scripting applications

6. The Runsql.exe program provides a simple interface to permit SQL queries to be executed on ODBC compatible databases, including the ability to incorporate the results of these queries into web pages.

7. The Runsql.exe program requires a configuration file to be created by the user, which specifies the DSN name, the username, password, and other information. More information on this program and this configuration file is provided in Appendix B of this distribution.

8. The Sigma SDK is available on request. Contact us for more information.

This section has provided only an overview of the API, which may be sufficient for most programmers to write their own application programs. Sigma is an open system, and other features of the API (not discussed here) Are available. On selected systems, the Sigma Framework SDK may include template files, additional documentation, and code stubs.

For more information on the Sigma API, contact the distributor.

Section 7: Framework Customization

This section discusses the customization features of the Sigma Framework, which permits users to change the look and feel of the program. This is an essential aspect of the Framework, mainly of interest to developers looking to adapt the system for OEM purposes, or to conform with corporate website standards.

- **Customizable HTML.** Sigma includes header and footer files, and other HTML files, which can be modified and tailored to display headers and footers on each page, or display content affecting look and feel.

- **CSS Style Sheet.** The generic Sigma Framework header references a style sheet that can be used to change various CSS styles, including fonts and colors.

- **Screen Colors.** Sigma permits modification of tab and screen colors, and includes other more obscure parameters that are useful for customizing the Sigma Framework appearance.

- **Customizable Install Files.** Sigma provides hooks into its install and uninstall dialogs, to permit users to customize these functions, especially useful in certain OEM applications.

The above features, in addition to the application features discussed in previous sections, combine to permit a high degree of customization. These customizable aspects of the Sigma Framework are introduced and discussed in the paragraphs that follow.

Header, Footer, And Helplink Files

One easy way to rapidly modify the Sigma Framework look and feel is to edit the header and footer files for the system, or the helplinks.html file, as follows:

- **.\s-html\s-header.html.** This file is displayed at the top of each page. By default, the header file that comes with the Sigma Framework is blank, only displaying a generic header, and referencing a generic style sheet.

- **.\s-html\s-footer.html.** This file is displayed at the bottom of each page as a last step on the page creation. By default, the footer file that comes with the Sigma Framework contains the copyright notice of the program. The user can add additional items to the footer, such as links to other locations.

- **.\s-html\helplinks.html.** This file is displayed to the right of the top-level navigation tabs, and can be used to display a link, a small image file, or other HTML. (In the case of the CorreLog Server, this image file executes the "More" dropdown menu.

Modification of the header and footer files, to add images, banners, links, special instructions, and even framesets is one of the fastest ways to make the program conform to a specific look and feel. Additionally the s-header.html file can reference JavaScript includes, special style sheets, and other essential components that are necessarily added to the <head> portion of an HTML document.

The header files are output using the SG_output_html() function of Sigma, so they can potentially contain Sigma macro definitions (as described previously.) The Sigma scheduler can rewrite the macro configuration files periodically to make these pages dynamic, if so desired.

The Sigma HTML Index File

When a user first accesses Sigma, the HTTP server reads the index file for the site as a standard operating procedure. The index file that comes with Sigma immediately redirects that user to the “/s-cgi/web.exe” program, which triggers HTTP authentication.

A developer can modify this behavior by editing the index file: CorreLog/s-html/index.html. Specifically, rather than redirecting the user to the “web.exe” program, the developer can add a link to this page, needed to access the web.exe program, along with other links to locations of interest within the enterprise.

Note that this behavior will be slightly different if the site server is IIS rather than the default Apache server. In this case, the index file is typically the file “default.htm”, and this is the particular file that should be edited by a developer.

In either case, the HTTP index file is available for modification and customization, and is typically replaced of modified in any OEM application, so as to conform to product or corporate specifications.

Sigma Error HTML Files

When specific errors occur on the system, error HTML files are displayed, residing in the “s-html” directory along with other HTML files on the system. These files can be replaced or branded, such as with personalized system contacts. The specific error HTML files are as follows:

- **CorreLog/s-html/error-access.html.** The “web.exe” program automatically displays this file if the user is not authorized to view a particular page (i.e. if the “SG_require_admin()” or “SG_deny_guest()” functions indicate that the S_ACCESS environmental variable is incompatible with the screen.

- **CorreLog/s-html/error-disabled.html**. The “web.exe” program automatically displays this file if the current user is disabled. The screen is displayed as soon as the user logs into Sigma, and cannot be bypassed. No screen can be accessed by a disabled user, other than this page.

- **CorreLog/s-html/error-empty.html.** The “web.exe” program automatically displays this file if a directory is found in the “sigma-web” directory that contains no files, i.e. is an empty directory. The user is notified that the specified application may have been uninstalled.

- **CorreLog/s-html/error-auth.html.** The Apache HTTP server automatically displays this file when an authentication error occurs, i.e. when a user login fails multiple times. This is configured in the “./apache/conf/httpd.conf” file.

- **CorreLog/s-html/error-file.html. The** Apache HTTP server automatically displays this file when an attempt is made to access a page that does not exist. This is configured in the “./apache/conf/httpd.conf” file.

The Web.exe program displays (and requires) the first three error HTML files. In contrast, the last two error HTML files are accessed by the HTTP server, and totally under the server control.

The Apache server permits the user to customize the screens associated with any error number that the server generates. This is documented in the Apache

manuals, available online and from a variety of sources. See the “CorreLog/apache/conf/srm.conf” configuration file for more information.

Sigma Welcome Screen

The first screen displayed when a user logs into the system is the “Home” screen, residing in “CorreLog/sigma-web/100_Home.html”. This is just an ordinary HTML file within the Sigma directory, and can be deleted or replaced with some other screen.

To display a different initial screen, or possibly access an initial application on login, the developer creates a file that is alphabetically less than any other file in the “sigma-web” directory, such as “001_Hello.html”. This leverages the standard capabilities of the Sigma Framework. In particular, it may be useful to keep track of user logins by creating an initial application or batch file, which records the user login time, and immediately redirects the user to some other screen.

Sigma CSS Style Sheet

The “s-html/s-header.html” file references a style sheet residing in the "CorreLog/s-html/styles.css” file. This is just an ordinary style sheet whose location is configurable in the s-header.html file. Like any other style sheet, the user can place global styles, such as fonts, colors, and other screen characteristics.

Style sheets (also referred to as Cascading Style Sheets or CSS) are content independent specifications, typically residing in web accessible locations, which are read by pages in order to define the layout and style of a web page. The separation of style from content provides a large amount of capability to experience developers who want to modify and manage the appearance of the Sigma Framework system.

This specification is formally documented in RFC 2318. Further discussion of style sheets is beyond the scope of this document. However, it is important to stress that customization of the Sigma style sheet provides significant control over the look and feel of the system, desirable in any OEM application, or when adapting Sigma to an enterprise.

By default, the Sigma Framework style sheet is very simple, and contains only those styles needed to set the font consistently within Sigma. The user may experiment with styles, including definition of new styles associated with their applications. Numerous references on CSS and style sheets is available on the web and from a variety of other sources.

Screen Colors

Navigation Tab colors are a single aspect of the system look and feel that is not governed by CSS. Specifically, the tab colors (and tab text) are controlled by the settings of the “Parms” screen, discussed in an early section of this manual. These colors, rather than being defined by CSS, reside in the “config/parms.cnf” configuration file of the Sigma root directory.

This permits application programs to access these colors more easily, so that they may reference these colors. Refer to the previous section, and a discussion of the “SG_get_tab_on_color()” and similar functions.

Note that the “Parms” screen, as well as other screens mentioned in earlier sections, may not be critical for the operation of the Sigma Framework. For example, a developer may simply disable the tabs for these screens by renaming (or deleting) the executable files from the “sigma-web” directory. This will hide the parameter editor, and whatever values reside in the “config/parms.cnf” file will be unchangeable except by logging onto the platform and editing the values via a text editor.

Customizable Install Files

As a final topic in customization, it is important to note that the install and uninstall programs, discussed in the very first section of this manual, are themselves customizable. Specifically, a developer can add the following two files to the “CorreLog/system” directory.

- **SG_INSTALL.bat.** This batch file, if it exists, is executed by the “CO-install.exe” program as a final step before displaying the “Finish” button. The batch file is executed as a background program (in an invisible console) and the installer waits for the program to completely finish before continuing to the final screen. This file can be used to launch another installer (via the Windows “start” batch file command) or can simply contain instructions and logic needed to perform a detail within the custom installation.

- **SG_UNINST.bat.** This batch file, if it exists, is executed by the “CO-uninst.exe” program AFTER the user clicks the “Finish” button. The batch file continues after the uninstaller exists, and can be used to delete all files from the installation, or perform other custom installation functions.

The above two “hooks” support the concept of the Sigma Framework as a platform for delivering OEM custom solutions, or for packaging the Sigma installation as part of a larger system.

OEM Customization

Note that, in many ways, Sigma Framework is ideal as OEM software. The system can be repackaged with customized software specific for an enterprise, and redistributed. This simply requires placing the custom files within the sigma-web directory, retaining other customized files, and then packaging the installation with WinZip or some other archiving utility.

Note that this is completely feasible due to the architecture of Sigma Framework, which is self-contained, and which does not rely on installing files in locations outside of the root Sigma directory.

The small footprint of Sigma Framework; the various customizable and extensible aspects of the program; the fact that Sigma is self-contained and does not scatter DLL or other files across the system; the ability to adapt the install and uninstall functions of the program – all of these features combine to make Sigma Framework an exceptionally good candidate to serve as a redistributable platform for OEM and corporate-wide development

Section Summary And Additional Notes

1. Sigma Framework permits customization at various levels, including HTML files, style sheets, screen colors, and extension of the Install and Uninstall programs.

2. Sigma can be repackaged with different screens. Existing screens can be re-arranged, disabled or deleted by simply re-arranging the contents of the “sigma-web” directory.

3. The initial Sigma login screen can be selected by placing a custom executable or file as the first file in the “sigma-web” directory.

4. The standard Sigma Framework application is quite simple and generic. For example, the entire system does not contain ONE SINGLE displayed image file. The Framework is designed to express the look and feel desired by a particular developer.

5. Sigma Framework is an ideal candidate for OEM applications, since the customization permits repackaging of the system with large amounts of custom software, specific for a particular application.

Section 8: Trouble Shooting & FAQs

This section finishes this manual by discussing common problems encountered by Sigma Framework administrators and developers, and how to troubleshoot and solve these problems.

The Sigma Framework is a very open system. Developers and administrators can modify it extensively, and creatively. As such, it is common to experience problems, especially when getting started. The range of the typical problems experienced is quite small, and can largely be addressed by the notes in this section.

Prior to opening a trouble ticket, users may wish to review the list of issues in this section. In addition to providing assistance with solving problems, this section can be used to achieve greater insight into the workings of the Sigma Framework. After reviewing this section, if you still have questions or issues, please contact customer support, or visit our website, or call us directly.

Thanks for your interest in the Sigma Framework. We are always interested in comments, and eager to discuss your particular applications!

CorreLog, Inc.
http://www.correlog.com

Q: How Do I Reset My Password File?

If you have access to the computer that is running the Sigma Framework, you can easily reset the default username and password back to username “admin” password “admin”. The default login database is installed in the “CorreLog/apache/install” directory, and these files can be manually restored using the following procedure:

1. Login to the platform running Sigma.
2. Copy the “CorreLog/apache/install/passwords.dat” file to the “CorreLog/apache/passwords.dat” file. (This is the default password file for the Apache HTTP server.)
3. Copy the “CorreLog/apache/install/pass.cnf” file to the “CorreLog/config/pass.cnf” file. (This is the default password file for the Sigma Framework.)

This will permit you to login to the system using the default username of “admin”, password “admin”. Note that this clears the password database back to its default value. All login information is lost and will not be recoverable.

Q: How Do I Manually Add Or Change Sigma Logins?

If you have access to the computer that is running the Sigma Framework, you can manually add logins. This is a bit more difficult than resetting the login database, but preserves the logins to the system. The procedure for manually adding a password is as follows:

1. Login to the platform running Sigma.
2. Change working directories to the “CorreLog/apache” directory.
3. Issue the following command at a command prompt, needed to add a new password to the “passwords.dat” file: “htpasswd.exe -b passwords.dat temp temp”. (This adds a username “temp”, password “temp” to the Apache “passwords.dat” file.)
4. Manually edit the “CorreLog/config/pass.cnf” file, such as with notepad.exe. The file contains encrypted login records, but you can add a record in clear text. Append to the file a record “temp temp admin none” (five words, space delimited). MAKE SURE you add a terminal new line to the new record. Save the file.
5. Login to the Sigma web interface with username “temp” password “temp”.
6. Go to the “System > Logins” screen and reset your original password, and optionally delete the “temp” username created above.

The “htpasswd.exe” program is the standard Apache utility for managing passwords on the system. Command line options for this utility are available by

executing the program with no arguments. More information on this utility is available from the Apache Group. (Consult the web or a variety of other sources.)

The “pass.cnf” file contains a list of encrypted passwords. Each record contains the username, password, access, and e-mail address of a user. Each encrypted record begins with a “//” character. When reading the pass.cnf file, if the record is not encrypted, the record is assumed to be a clear text representation, and the program continues. Next time that the pass.cnf file is modified via the Sigma “System > Logins” screen all the passwords are re-encrypted.

Q: What Is The “Sigma-cgi\Htaccess.txt” File?

The “htaccess.txt” file contains a pointer to the password file. Users can copy this file to different directories to enforce a login prompt when that directory is first accessed. This file is first configured when Sigma is installed, and contains the full pathname to the passwords.dat file. If this file is tampered with, or corrupt, users will not be able to login to the Sigma system. In this case, the administrator must reconfigure the system. An experienced administrator can simply edit the htaccess.txt file. You can also just execute the installation procedure “system/CO-install.exe”, which will reset all the passwords back to the default username “admin”, password “admin”.

Q: Why Don’t Files And Directories Appear As Navigation Tabs?

If you copy files into the “sigma-web” directory, and they do not appear as navigation tabs, the most likely reason is that you have failed to follow the strict naming conventions required by Sigma Framework for naming files in this directory. Sigma silently skips over any file in the “sigma-web” directory that does not following the precise naming conventions.

Each object in the “sigma-web” directory needs to have a three digit number (with possible leading zeros) followed by an underscore character, followed by the tab name, followed by a suffix of either “dir”, “txt”, “htm”, “html”, “cmd”, “bat”, or “exe”, as documented in Section 5 of this manual. Any file that does not follow this precise naming convention is ignored by the “web.exe” program.

Verify that you have precisely followed these naming conventions. Further verify that the file suffix values actually represent the nature of the file (for example, verify that any object with a “dir” suffix is actually a directory.) This is an easy mistake to make. Because the Sigma names are slightly complex, you can very easily forget to prefix an object with exactly three characters, or leave off a “dir” suffix from a directory.

Note that file and folder permission issues, while often suspected as the cause for this problem, are rarely the actual problem. The HTTP server runs as the “Local System” user, which generally has access to all files and folders on the system.

Q: HTTP Server Is Not Responding. How Do I Fix This?

The most likely reasons for this occurring are that the CO-Apache.exe program is not running, or running at a port number different from that which is expected, or that the machine running Sigma is not accessible to the network, or a problem with an Apache configuration file exists. There is a methodical way to troubleshoot this problem as follows:

1. Login to the platform that is running the Sigma Framework.
2. Using the Windows Task Manager, confirm that the CO-Svc.exe program is running. If this program is not running, it should be started using the Windows Service Manager, or via the “net start sigma” command at a command prompt.
3. Using the Windows Task Manager, confirm that the CO-Apache.exe program is running. (There should be precisely two instances of this program running on the platform)
4. If the “CO-Apache.exe” program is not running, but the “CO-svc.exe” program is running, then the “CO-Apache.exe” program may have stopped due to an internal Apache error. Try starting Apache manually. At a command prompt, changing working directories to the “CorreLog/apache” directory and execute the command: “CO-Apache.exe”. Observe any error messages, and take corrective action.
5. If the “CO-Apache.exe” program is running, but is still not accessible, then the Apache server it may be listening at a port number that is unexpected. Check the “conf/apache/httpd.conf” file to see what port number Apache is listening to (i.e. check the value of the “Port” entry in the file.) Then try accessing the Apache server by incorporating that port number into the URL, such as “http://localhost:88”.
6. If you are able to access the Apache server from the localhost platform, but not from some other program, check to make sure that the Proxy settings of your browser permit access to the local area network. On Internet Explorer, you can access “Tools > Internet Options > Connections > Lan Settings” to view or modify your Proxy settings.

Note that Virus Protection programs and port blocking security software can interfere with the startup of the Apache server, and interfere with the installation procedure of the Sigma Framework. If you are running the McAfee virus protection program (or some other port blocking software), temporarily disable that program to see if it corrects the problem.

Finally, note that the “netstat –a –p tcp” command, issued at a command prompt, will list the current TCP service ports that are open on your platform. You should verify that the service port used by Apache (that is configured in the “apache/conf/httpd.conf” file) is busy ONLY when the CO-Apache.exe program is running.

Q: Why Does My Program Work Outside Of Sigma, But Not When Installed?

The most likely reason that your program works outside of Sigma (such as at a command prompt) but fails to work correctly within Sigma is that you have programmed some dependency that exists only in your programming environment.

Note that the working directory for all programs executed by Sigma is the “CorreLog/s-cgi” directory. All pathnames used by a program should be relative to that directory. Further note that Sigma executes programs as the “Local System” user, and inherits that environment.

A simple way of getting the environment used by Sigma programs is to create a simple test batch file, called “999_Environment.bat”, and copy that program into the “sigma-web” directory. In that batch file, issue the single “SET” command, which will list all the environmental variables of the program. (Also, you can configure other environmental tests.) When you access the “Environment” tab, you will see all the environmental variables that are present.

Finally, you can always include debug statements in a program, and write debug data to a log file as a CGI program executes. This provides a simple way to trace execution of a program.

Q: How Do I Decrypt Sigma Data? (Cannot Find “decrypt” Function.)

Sigma does not provide access to any decryption function. The Sigma encryption service is one way. The actual encryption algorithm uses a block, rotating, time sensitive cipher algorithm, which is highly secure. Once the data is encrypted, there is no obvious way to decrypt the data outside of Sigma.

Note that Sigma is designed so that it is never necessary to actually decrypt the data components. The user encrypts URL arguments. The “web.exe” program, prior to executing any user programs, decrypts these arguments. The user receives the unencrypted data as command arguments.

Finally note that the password file is double encrypted by different methods, so it is even more secure that the URL encryption, and cannot be subverted (such as by attempting to pass the login records as arguments to user defined scripts.)

As with any security system, it may be possible for an expert programmer to decode or decrypt the encrypted data (such as by editing the Web.exe program with a binary editor, to extract the decryption method employed.) Any Attempts to do this is, or to subvert the encryption process, is a DIRECT VIOLATION of the Sigma license, agreed to by your organization. Consult the LICENSE.txt file for more information.

Q: Why Can't I Pass Arguments To My CGI program?

The answer to this question depends upon whether you are encrypting the URL or not, because the two mechanisms for passing arguments via a URL depend upon this.

1. If you are not encrypting the URL, then command arguments are passed using the standard "&" character to separate the arguments. Also, you must be very careful not to introduce special characters into the arguments, such as unquoted spaces, or an equal sign. (This may entail a good deal of lookup of HTML character codes!)
2. If you are encrypting the URL, command arguments must be appended to the pathname of the executed program, separated by semicolon ':' characters. The user must quote any semicolon in a passed argument with a backslash. The data does not have to otherwise be formatted.

The "010_DisplayArgs.bat" file, shown in Section 5 of this manual, may be useful in debugging this problem.

Additionally, you can view the source HTML document (via the normal browser "View Source" option) to see the command arguments that are launched with any subprogram. (This is an implicit function of the "SG_run_command()" function, discussed in Section 6 of this manual.) Whenever the "web.exe" program or any subprogram executes another program using this function, the decoded command and arguments are incorporated into the source HTML as commentary.

Q: How to I Incorporate PHP, Perl, Ruby Into Sigma?

Sigma supports any programming language, including all of the above, plus others, such as Java. The user must create a wrapper ".cmd" file, which launches the interpreter for the language and references the script. This allows a user, in addition to launching an interpreter, to also have full control of the environment in which the interpreter is launched, including the setup of any environmental variables, working directories, permissions, etc.

This technique is extremely straightforward. Some users might object to the fact that their scripting language needs to have a batch file wrapper, but it actually provides a vast amount of flexibility that often doesn't exist in other frameworks. Through the enforcement of this simple rule, a vast amount of extensibility is made available that is otherwise extremely problematic to programmers.

Note that the batch file can consist of a single line. (Also recall that, in a batch file, you can turn off echoing of command output by starting the program with the command "@echo off", which inhibits command line echoing.) Also note that arguments from the batch file can be passed to the third party programming language using the "%1%", "%2%" variables.

A discussion of batch file programming is beyond the scope of the question, but it should be stated that most programmers are unaware of how powerful batch files may be, especially in later versions of the Windows operating system. Consult the web for information on batch file programming, or seek information from a multitude of other sources.

Q: Why Is My Script Displaying Pre-formatted Text Rather Than HTML?

If you are launching Perl, Ruby, Java, or any other programming language within Sigma, you must use a batch file that begins with a "cmd" extension AND NOT a "bat" extension. Sigma Framework automatically displays output of a ".bat" file as preformatted text (without HTML). The "cmd" extension is used to display raw HTML output. This is the only difference within Sigma between these two file extensions.

Generally a ".bat" file is mainly useful (over a ".cmd" file) in order to display raw text files, or execute system programs such as "netstat", which deliver preformatted text as a standard part of their operation. For example, the user can create an "800_SysTools.dir/100_Netstat.bat" file wrapper, and an "800_SysTools.dir/200_IpConfig.bat" file wrapper, and other screens which run system commands. This technique can be used to generate a variety of simple to program screens that are useful for system management.

Q. Why Are My Navigation Tab Colors Messed Up*?*

The Navigation Tab Colors are set by the System > Parms screen, and the data resides in the "./config/sparms.cnf" file of the Sigma Framework. The colors must either be in RRGGBB hexadecimal notation (which is the standard HTML format) or must be one of the HTML predefined color names such as "Red", "Blue", "Azure", "Burlywood", "Navy", etc. (A vast variety of predefined color names are supported by all browsers.)

The most likely reason your colors are unexpected is that you accidentally introduced an errant character in the tab color while editing the “Parms” screen. You can set the colors back to default (along with all other parameters) by accessing the System > Parms screen, and then clicking the “Default” button on the “Edit” screen. Or, you can simply edit the “./config/sparms.cnf” file with a text editor and carefully make color changes.

Note that the text color and the field color should be selected for a good contrast. Generally, users may want to confine the text colors to be either WHITE or BLACK for the best contrast with any Tab color field.

A developer can remove or rename the Parms screen by accessing the “sigma-web” directory. The screen is not required for normal operation of the Sigma Framework, and is provided as a core application only for user convenience. Rather than using this screen, the administrator can simply edit the “config/sparms.cnf” file with a standard text editor, such as “Notepad”.

Q: How Many Navigation Tabs Can I Create?

A typical system might have hundreds, or even thousands of nested tabs. Consider that a system with five tabs at each of three levels will have 125 screens. A system with eight tabs at four levels will have 4096 different screens.

At any given level, the maximum number of tabs is actually limited to five hundred per level. However, this will cause a horizontal scroll bar to be opened, and make navigation of the system very difficult. In practice, depending upon the size of each tab name, a typical level of navigation might have between one and ten different tabs.

As with any system, the organization of the data (in terms of directories and subdirectories) is critical to the simplicity of the system. It is fairly easy to re-arrange the programs and navigation tabs within Sigma using the Windows File System Explorer or other technique. Developers should exercise moderate design deliberation needed to create an ergonomic and logical arrangement of tabs.

Q: Why Do I Keep Getting “Disabled User” Messages?

The “disabled user” message is displayed under two different circumstances, as follows:

1. The user is actually disabled in the System > Logins screen. This is a specific setting that Administrators can use in order to prevent a particular user from accessing the system.

2. The user is defined in the HTTP server, but not in Sigma, and the default user setting is set to “disabled”. This is a way of enabling Windows Directory Services as the means of authentication, but disabling any user that is not specifically registered with Sigma.

If you are using Windows Directory Services, and you wish all users to be able to access Sigma Framework programs as “Guests”, you can set the default user, in the “System > Parms” screen to be “guest”. Then, if the user is registered on the network, that user can at least view some of the Sigma screens. In that case, only those users who are registered as “disabled” will receive “disabled user” screens.

This method provides a good deal of flexibility in who can access Sigma Framework, especially in the maintenance of passwords. For example, you can use Windows Authentication (with IIS) and limit only those users registered with the program to access screens. When users change their passwords on the network, access to Sigma Framework changes correspondingly without the need for changing passwords within Sigma.

Note that when using IIS, or some other server making use of Windows Directory Services, the passwords configured in Sigma are essentially ignored. The Sigma passwords are pertinent only when using the default configuration of Sigma and the Apache HTTP server.

Q: Why Is Sigma Is Not Working With Microsoft IIS?

Normally, Sigma works perfectly with IIS. However, there are a variety of arcane permission problems that can prevent easy integration of Sigma Framework and IIS, depending upon the security policies implemented on the host platform and network. Some of these problems may require intervention by Technical Support in order to fix (such as using Sigma with “hardened” versions of IIS.)

The most obvious and common reason that Sigma is not working with IIS is that the program “virtual directories” have not been correctly configured. Verify that the “CorreLog/s-cgi” directory folder corresponds to the “/s-cgi” URL, and that this URL provides “read” and “execute” permissions. Also verify that the “CorreLog/s-html” directory folder corresponds to the “/s-html” URL, and that this URL provides at least “read” permissions.

The “CorreLog/sigma-web” and other folders within Sigma Framework SHOULD NOT normally correspond to URLs on the system, to prevent unauthorized viewing of files and executables. (In some situations, where security is not an eminent concern, these folders may be made public, depending upon the goals of a developer.) Consult IIS documentation on security policies for more information.

Q: Can Sigma Authenticate Users By Group?

Normally, Sigma authenticates users only by username. However, using the IFMEMBER.exe utility that comes as a standard component of the Windows SDK, it is possible to authenticate users by group rather than by name.

For example, it is possible to grant “admin” rights to any user in the “Admin” group, and grant “guest” rights to any user in the “Design” group, and deny all other users on the network. This implementation may require the help of Technical Support. Contact us for more information on this particular implementation.

Q: Where Are The Development Libraries, Sigma.dll?

Many Sigma Framework evaluation versions (in particular those versions that have been downloaded from the Web) omit these objects from the distribution. Developers interested in the Sigma Framework Development SDK should contact us.

If you have installed these components, you will find them in the “CorreLog/SDK” directory of the site. This directory furnishes link libraries, include files, the Sigma.dll, coding templates, and additional documentation.

If the Sigma SDK does not exist at your site, these items can be installed in the system without affecting other components.

Note that without these components, developers can still create applications using their favorite programming and scripting languages. The “sigcmd.exe” utility is particularly useful and essential for these programming efforts, and this utility is a standard part of ALL Sigma Framework distributions, located in the “CorreLog/system” directory. See Section 6 of this manual for information on how to use the “sigcmd.exe” program.

Q: How Do I Move Sigma To A New Machine / Location?

Because Sigma Framework does not scatter DLLs or other system files across the platform, it can be easily relocated. Simply copy the Sigma directory to its new location. (You can also Zip Sigma and Unzip at to its new location.) Once the directory is copied, the administrator can manually execute the “system/CO-install.exe” program to reconfigure the Apache server to work with the new path.

1. Stop the Sigma Framework Server using the Windows Service Manager, or by executing the “net stop sigma” command at a command prompt.
2. Save the “config/pass.cnf” and “apache/passwords.dat” files. (They will be used in step 5 below.)

3. Copy the “Sigma” root directory to the new location. This can be a new disk on the system, or a different machine on the network.
4. At the new location, run the CorreLog/system/CO-install.exe program, and follow the steps needed to finish the install procedure.
5. Copy the config/pass.cnf and apache/passwords.dat files, preserved in step 2 above, into their new locations.

As a final step, you can optionally uninstall the old location using the standard Windows Add / Remove Programs facility.

Note that the above steps are sufficient for re-installing the Sigma Framework core files. Specific applications within the Framework may require special procedures, depending upon their functions and design. Although the above steps may be sufficient for moving these applications, the documentation associated with these applications should be consulted for details.

Q: Why Didn’t The Uninstall Program Remove All The Sigma Framework Files?

The final cleanup of Sigma files is left as an obligation to the user. This is made easy by the fact that Sigma does not scatter DLLs or affect system files, hence the uninstaller can simply drag the Sigma root directory to the Windows “Recycle Bin” in order to dispose of these files.

The “system/CO-uninst.exe” program (which is normally executed by the Windows “Add / Remove Programs” facility, but which can also be executed manually) removes the Sigma service and cleans up the registry. It does not remove any files on the disk. Due to the simplicity offered by the total self-containment of Sigma, the final removal of these files is left as a simple manual step. This enhances data security, and provides substantial resistance to accidental deinstallation of important data.

Q: What Sigma Files Should I Periodically Backup?

The details associated with safeguarding system data depend upon the particular applications that reside in Sigma. Generally, the only data that is specific to the core Sigma Framework is found in the few files within the “./CorreLog/config” directory. If these files are periodically archived, they can be restored in order to preserve the Sigma usernames, scheduler information, screen colors, and other configuration items.

Applications within the Framework may require special backup procedures, depending upon their functions and design. The documentation associated with these applications should be consulted for details.

Q: How Do I Schedule A Program To Run Each Minute?

The scheduler program permits files to be run hourly, daily, weekly, and monthly, but not each minute. If a dedicated process is required to accomplish this, you can create a script or batch file that loops, executing commands, and sleeping for 60 seconds. This program can then be launched via the Sigma Scheduler “start” directive, on program startup, where it will run as a persistent program on the system.

The Sigma scheduler function permits any process to be started as a service. The program is terminated automatically when the system stops. This affords a mechanism to run any persistent process in background, including batch files or scripts that perform periodic action such as that described above.

Q: Why Can’t I See My Scheduled Batch File Running In The Windows Task Manager?

You can run batch files without modification or special regard using the Sigma Scheduler interface, however the name of the batch file will not be visible during execution. This is because the scheduler launches the CMD.EXE program to execute the batch file, and that is the name you will see in the Windows Task Manager. It may therefore be a good idea to provide some error and status logging within the batch file to ensure it is actually running.

Q: Where Are Sigma Errors Logged?

The only log file generated by the core Sigma Framework is the Sigma Service program log, which “system/CO-svc.log” file. This file contains an indication of processes that have been started, processes that have unexpectedly terminated, and any messages due to the “log” directive of the Sigma Scheduler. The file is automatically deleted and restarted each time the CO-svc.exe program starts.

The Apache server provides an extensive logging facility, detailed by the Apache group. By default, the CO-Apache.exe program creates the “apache/logs/error.log” file, containing any errors experienced by the Apache server. Users can also log access, and many other indications. Consult the Apache Group website for details.

Applications within the Framework may have special logging capabilities depending upon their functions and design. The documentation associated with these applications should be consulted for details.

Q: How Do I Put A Space In A Tab Name, Without Wrapping The Text?

Files in the “sigma-web” directory should contain spaces. (The closest equivalent would be an underscore or perhaps a hyphen.) This constraint enforces uniformity between Sigma applications. For consistency, Sigma developers should select names for their programs (and their corresponding tabs) that do not contain spaces. If a program contains a space as part of its name, the program name will wrap, increasing the height of the tab beyond normal. This may affect other aspects of the display as well.

Q: Why Are Links In My Web Pages Broken In Sigma?

The most likely reason for this is that the web page uses “relative” naming in hyperlinks, as opposed to “absolute” paths. Because the “web.exe” launches programs, all relative paths are with respect to the “/s-cgi/” directory. This may have negative consequences to existing documents that are moved into Sigma. It may entail some modification of these documents..

One way to ensure that URLs work is to make all URL paths absolute. For example, the link “<img source=images/my.gif>” can be converted manually with an editor to be “<img source=/images/my.gif> to correct this problem. (Note the forward slash, added to the “source=” attribute.) This decouples the link from the base URL of the document, and insures the proper image is accessed.

Rather than editing many different documents and URLs, developers can employ the HTML <BASE> tag, in the “s-html/s-header.html” file, to provide a base URL for all other URLs in the system. This is a simple way of correcting this problem, but may have undesirable side effects associated with some user supplied documents, depending upon the particular applications running in the framework. Developers should try this, to see if it is a good solution.

Q: How Does A Script Get The Tab Colors?

The tab colors are stored in the “CorreLog/config/sparms.cnf” file, and can be parsed by application programs and scripts. If a developer is using the “Sigma.dll” and Sigma SDK, there are various methods (SG_get_tab_on_color(), for example) that can be used to get the configured colors. Otherwise, the application program can parse the colors, or any other value, from the configuration files.

All configuration files within Sigma Framework use the exact same standard way of representing configuration data, which makes this activity somewhat easy. All configuration data consists of a unique keyword identifier, followed by one or more white spaces, followed by a value.

Q: How Do You Change The Service Port For Apache?

Sigma employs a standard version of Apache, at least with respect to all configuration items and values. The Apache TCP port number is specified in the "apache/conf/httpd.conf" file, identified by the "Port" directive. Users can stop the Sigma Framework Service, edit this file to provide a new value, and restart the service. This will change the TCP port number at which Apache listens for service requests.

The configuration data associated with Apache is quite extensive, and is beyond the scope of this question. The configuration files that come with Sigma contain only a very small subset of the total directives available to users. In some ways, this helps simplify the maintenance of the Apache server, and is a reasonable implementation. More advanced directives can be added by experienced users.

Q: Can The Sigma Installer Run A Special Program?

The Sigma Framework installer, residing in the "system\CO-install.exe" file, provides a method for executing extra commands. Developers can create the "system\CO-INSTALL.bat" file, which is automatically executed by the installer as a last step in the installation program. This provides a method for launching other installers, and performing extra installation work that may be required to support a particular Sigma application.

To launch another program, the developer should make use of the standard "start" batch command, which allows foreground processes to be opened from background processes. (The CO-INSTALL.bat file normally runs in an invisible console, hence cannot prompt for input.) For assistance, type "start /?" at a command prompt.

Note that there is also a facility to run special uninstall commands. When executing the system\CO-uninst.exe program, after the user clicks the "Finish" button, the system\CO-UNINST.bat file will be executed (if it exists). The process will continue after the uninstaller exits, and can be used to perform additional deinstallation work for a developer.

Q: Sigma Screens Have Suddenly Stopped Accepting Post Requests. What happened?

There is only one likely reason for this happening: The disk that is executing the Sigma Framework is out of space. When posting arguments, a short file is created in the "CorreLog/temp" directory. If the file cannot be created, then form posts will stop working. (Typically, buttons will have no action except to redisplay the screen, although this depends upon the application program.

Appendix A: List Of Framework Files

This appendix provides a comprehensive list of all Sigma Framework Files and Folders that come with the core distribution. Framework developers will find this a useful reference and explanation. The section is mainly of interest to application developers, or administrators.

./LICENSE.txt

The license file for Sigma, displayed via the “View License” button on the startup dialog. (Please read.)

./apache

The directory containing all Apache files on the system.

./apache/ApacheCore.dll

Required DLL for the Apache server.

./apache/htpasswd.exe

A command line utility, used to maintaining the Apache server passwords. This program is executed via the Sigma “Login” screen, and can also be executed manually by experienced administrators. Type “htpasswd.exe -?” form brief built in help.

./apache/httpd.pid

The Program ID for the currently executing Apache program. If this file does not exist, then the Apache server is not running. (However, existence of the file cannot be used to determine whether the server is actually running.)

./apache/passwords.dat

The encrypted passwords of the system, maintained by the "htpasswd.exe" program, and by the Sigma "Login" screen.

./apache/CO-apache.exe

This is the actual Sigma Apache server executable. This is a standard version of the server, but slightly hardened and cleaned up for the Sigma server. The server name is prefixed with "SG" so that it appears clearly in the Windows Task Manager.

./apache/Win9xConHook.dll

Required DLL for the Apache server.

./apache/conf

The directory that contains Apache server configuration files. This directory is required for proper operation of the server.

./apache/conf/access.conf

Apache configuration file, specifying the access and permissions to various functions and features. While these directives commonly exist as part of the httpd.conf file, the Sigma distribution splits these directives apart into this separate file.

./apache/conf/httpd.conf

Apache configuration file, specifying the server name, and port number, and containing pointers to other locations within the Apache system. This is the main configuration file for the apache server, and can be heavily customized by experienced developers.

./apache/conf/mime.types

Apache configuration file, specifying the mime types that are supported by the Sigma Apache server. This file contains a list of mime headers and file extensions, used by client programs to determine how to display non-HTML data. The file contains only a brief list, to enhance security. Developers may wish to load more mime types.

./apache/conf/README.txt

A README.txt file that explains the purposes of the files in this directory.

./apache/conf/srm.conf

Apache configuration file, specifying the system resources of Apache. In particular, this is the file that performs the mapping of virtual URL folders to the actual disk folders on the system. While these directives commonly exist as part of the httpd.conf file, the Sigma distribution splits these directives apart into this separate file.

./apache/install

Directory containing template configuration files, used in the installation process. These files should not be touched or modified by users. They exist only to support the CO-install.exe program.

./apache/install/access.conf

Template access.conf file, used only during installation.

./apache/install/htaccess.txt

Template htaccess.txt file, used only during installation.

./apache/install/httpd.conf

Template master httpd configuration file, used only during installation.

./apache/install/mime.types

Template mime.types file, used only during installation.

./apache/install/pass.cnf

Template Sigma password file, used only during installation.

./apache/install/passwords.dat

Template Apache password file, used only during installation.

./apache/install/README.txt

A README.txt file that explains the purposes of the files in this directory.

./apache/install/srm.conf

Template srm.conf file, used only during installation.

./apache/logs

This directory contains log files generated by the system, as configured in the Apache “httpd.conf” file. By default, this directory will contain files ONLY if the Apache server experiences an error (which will create the “error.log” file.) Users can edit the httpd.conf file to enable transfer logging, which logs the visit of all users to the website. Additional logging capability is also available.

./apache/logs/error.log

This is the standard error log for Apache. It may or may not exist, depending upon whether the Apache server has experienced any error. Note that the existence of this file does not indicate a critical problem. (All errors, including accessing invalid URLs by users, are logged, and the messages may merely be informative rather than actual error conditions.

./apache/logs/README.txt

A README.txt file that explains the purpose of the files in this directory.

./config

This is the main configuration folder for the Sigma system. By default, the directory contains only those files needed to support the login mechanism, the scheduler function, and system parameters. Application programs may add their own unique files to this directory.

./config/pass.cnf

File containing the encrypted passwords for Sigma logins.

./config/sched.cnf

File containing the scheduler commands, read by the persistently running CO-svc.exe program, used to launch processes on startup, shutdown, or at scheduled intervals.

./config/sparms.cnf

File containing the system parameters of the system, maintained by the Sigma “Parms” screen.

./s-cgi

This folder corresponds to the /s-cgi URL, and the contents of this folder consists of programs that can be executed by users. Special care should be taken to prevent unauthorized access to this folder, since malicious users could exploit this directory by copying in their own programs for remote execution.

./s-cgi/htaccess.txt

File containing the access permissions to this folder. This file is required in order to get a login prompt. The file name is configured in the “srm.conf” file, and the file contains directives that prompt a user for login before accessing any file in the directory in which it resides. Users can copy this exact htaccess.txt file to any other location in Sigma to force a login prompt.

/s-cgi/web.cnf

This configuration file is used by the “web.exe” CGI program, and specifies some miscellaneous details regarding the web display, such as the size of the tab buttons, and the default tab colors, as well as certain path locations. Users should generally not modify this file. If it becomes corrupt, the user can delete the file and the "web.exe" program will recreate it, with defaults, the next time the program is executed.

./s-cgi/web.exe

This is the central CGI executable program, referenced each time a file is accessed within Sigma. The program creates the navigation tabs, and manages the execution of other programs.

./s-doc

This directory contains miscellaneous documentation. It is viewable via the Web interface, as configured in the srm.conf file. (You can access this URL via the “/s-doc” URL.)

./s-doc/FW-MANUAL.pdf

The Sigma Framework Users Manual, in Adobe PDF format. This manual can be accessed via the Welcome screen.

./s-doc/HEADER.html

A header file, which puts a banner at the top of the screen when the directory is viewed directly at the /s-doc URL. (This file exists many for aesthetics.) See the srm.conf file.

./s-doc/LICENSE.pdf

The Sigma Framework Licensing Agreement, in Adobe PDF format. This is identical to the LICENSE.txt file in the root directory of Sigma.

./s-html

This directory contains HTML files, in particular error html files, but also the main index for the web browser. This directory corresponds to the /s-html URL, as configured in the apache/conf/srm.conf file.

./s-html/error-access.html

HTML file, displayed to a client whenever an access error occurs, i.e. if a “guest” login tries to access a screen that requires and “admin” login.

./s-html/error-auth.html

HTML file, displayed to a client whenever an authentication error occurs, where the user fails multiple times to login to Sigma. This file is accessed directly by the Apache server, as configured in the apache/conf/srm.conf file. (This is a 401 error.)

./s-html/error-disabled.html

HTML file, displayed to a client whenever a disabled user logs into the system.

./s-html/error-empty.html

HTML file, displayed to a client when a tab is accessed, and the tab corresponds to a directory in the “sigma-web” folder that contains no files. (The administrator should probably delete that directory.)

./s-html/error-file.html

HTML file, displayed to a client whenever an attempt is made to access a file directly by a URL, and the file does not exist. This file is accessed directly by the Apache server, as configured in the apache/conf/srm.conf file (This is a 404 error.)

./s-html/index.html

The main index file for the Sigma implementation of the Apache server. This is the first file accessed whenever a user hits the server. The file simply redirects a user to the “/s-cgi/web.exe” program, which causes a prompt for a login. This file can be tailored by a developer, such as to add links to other locations of interest, including the login screen.

./s-html/s-footer.html

HTML file, displayed at the bottom of each screen by the web.exe program after all other programs are launched. This provides a way to customize the display with certain links, a copyright notice, etc.

./s-html/s-header.html

HTML file, displayed at the top of each screen by the web.exe program prior to generating any other output. This is an important file, because it can include JavaScript, resources, select style sheets, as well as display a banner image and hyperlinks. The generic implementation is quite simple, and can be customized extensively.

./s-html/helplinks.html

HTML file, displayed to the right of the top-level navigation links. This file runs the "More" tab in the CorreLog Server system, but can be modified to display other links, images, or arbitrary HTML.

./s-html/styles.css

The Sigma “Cascading Style Sheet”, specified in the “s-header.html” file. This is a simple file that can be expanded upon by a developer to change the look and feel of the program, including fonts, colors, borders, and special effects.

./s-html/images

This directory contains image files used by the system. This directory corresponds to the /s-html URL, as configured in the apache/conf/srm.conf file. The directory is viewable at the “/s-html/images” URL. (The administrator or a developer may wish to place an index file in this directory to hide it from prying eyes, if that is a concern.)

./s-html/images/HEADER.html

A header file, which puts a banner at the top of the screen when the directory is viewed directly at the /s-html/images URL. (See the srm.conf file.)

./s-html/images/folder.gif

A small image of a folder, used in directory listings.

./s-html/images/note.gif

A small image of a note, used in directory listings.

./s-html/images/sig-logo.gif

A logo file for the Sigma system.

./s-html/images/sxii.jpg

The banner for the corporate website.

./sigma-web

This is the all-significant “sigma-web” directory, which is the root directory used by the “web.exe” program. Within this directory you will find folders and files that follow a specific naming convention, used to create tabs and provide automatic navigation features. See Sections 1, 3, 4, and 5 of the manual. Note that this directory is not visible directly with a Web browser, i.e. the /sigma-web” URL should yield a 404 Page not found” error.

./sigma-web/100_Home.html

This is the welcome screen, which is the very first screen viewed upon successful login to the system.

./sigma-web/900_System.dir

This directory corresponds to the “System” tab, displayed at the top of the screen to the right of the “Welcome” tab.

./sigma-web/900_System.dir/100_Logins.exe

This is the “Logins” screen, which is the first screen viewed upon clicking the “System” tab. The program permits users to add, modify, and delete logins from the system.

./sigma-web/900_System.dir/150_ODBC.exe

This is the "ODBC" screen, which permits the user to configure Windows DSN (Data Source Name) values such as usernames and passwords. The screen also permits the user to run simple SQL statements.

./sigma-web/900_System.dir/200_Parms.exe

This is the “Parms” screen, displayed to the right of the “Logins” tab. The program permits users to edit parameters contained in the “./config/sparms.cnf” file of the system.

./sigma-web/900_System.dir/300_Schedule.exe

This is the “Schedule” screen, displayed to the right of the “Parms” tab. The program permits users to specify programs that are started, stopped, and which are run at periodic intervals by the CO-svc.exe program. The program edits the “config/sched.cnf” file of the system.

./sigma-web/Util

This directory contains miscellaneous utilities that are launched by the other web programs. These utilities generally create a single tab, nested in the screen, and are launched by command line arguments.

./sigma-web/Util/wusys.exe

This is the "System Info" utility, launched via a hyperlink at the bottom of the "Parms" screen, which displays information on the current system configuration.

./system

This directory contains executable programs, and system components. This directory is in the search path of all programs launched by the “web.exe” program, hence is a useful location to put programs that are required by a user application. (That way, the developer does not have to specify the full path to the application program.)

./system/CO-install.exe

The Sigma installer program. This program is launched automatically after extracting Sigma files. It can be run after installation to reset configuration data and reconfigure the Apache server. (See Section 2 of the manual.)

./system/CO-svc.exe

The Sigma Service Manager / Scheduler program. This program is directly controlled by the Windows Service Manager entry, and is responsible for scheduling activities on the system, including starting and stopping the Sigma service. The program reads the “config/sched.cnf” file, which is edited by the “Schedule” program available under the “System” tab.

./system/CO-svc.log

The log file for the CO-svc.exe program above, containing any errors encountered by the system, and a list of processes that have been started. The file is overwritten each time the program starts (so does not require administrative maintenance.) It is provided mainly for system debug.

./system/CO-uninst.exe

The Sigma uninstaller program. This program is launched by the Windows “Add/Remove Programs” facility. It can be run manually to remove the registry items for Sigma, and uninstall the Sigma Framework Service. The program does not delete any files. (See Section 2 of the manual.)

./system/sigcmd.exe

A utility for outputting encrypted URLs, and performing other activities. This program is documented in Sections 5 and 6 of this manual.

./system/sigma.dll

The Sigma DLL. This program is not actually required or used by any process in the core Sigma Framework system. (The core processes are statically linked) It is provided to complete the Sigma Framework API, used by developers to add their own applications to the system.

./temp

A temporary directory, required for the proper operation of Sigma. This directory MUST exist, and permit programs to write to the directory. Otherwise, side effects such as being unable to post arguments, will occur. See Section 8 of the manual for more information.

Appendix B: Sigma SQL Interface

This appendix provides a description of the Sigma SQL Interface, which is a simple but comprehensive mechanism for executing SQL statements on ODBC compatible databases, and optionally displaying the results of these queries in Web pages.

The RunSQL.exe program resides in the "system" directory of all Sigma Framework implementations. The program accepts as its first argument the name of a configuration file, whose contents are documented here.

The RunSQL.exe program can be executed interactively, at a command prompt, but is typically executed by either the Sigma "Macro" capability, or by a ".cmd" file. The user can define the SQL query to be executed within the configuration file, or can specify the query on the command line, after the configuration file pathname.

Prior to executing the RunSQL.exe program, the user must first configure an ODBC "Data Source Name" (DSN) in the "Control Panel > Admin Tools > Data Sources" dialog of the Windows platform. (This will require the user to log into the host Windows platform with an Administrator type login.)

Once a DSN is defined, the DSN name is specified in the configuration file, along with other directives, such as a "username", "password", "usedb", "maxresults", "listfmt", and "sqlstmt" values, described here.

If the RunSQL.exe program generates output (such as by running an SQL "select" statement) this output can be formatted as text, raw html, or xml, to support various implementation strategies.

Configuration File Description

The RunSQL.exe program requires creation of a standard Sigma configuration file, consisting of a series of directive keywords followed by the keyword values. (This is the same configuration file format used in the "Macro" facility, but is not otherwise related to Sigma Macros.)

There can be many different configuration files, each containing different queries and DSN names. Since the configuration file is specified as the first command line argument to the RunSQL.exe program, it is common for a complex system to contain dozens of configuration files, each designed for a specific purpose and corresponding web page.

The basic RunSQL.exe configuration file is depicted below:

```
dsnname             Sigma Access Database
username            None
password            None
usedb               None
maxresults          50
sqlstmt             select * from mytable
listfmt             html
```

The above configuration file contains all supported RunSQL.exe directives, and can be located anywhere on the system. The file, by convention, has a ".cnf" extension, but the program enforces no particular naming conventions. The pathname to this file is passed as the first argument to the RunSQL.exe program, and may use either forward or backward slashes.

Each directive in the configuration file is documented as follows:

dsnname

Required. This is the name of a "System" ODBC Data Source Name, configured in the windows platform "Control Panel". The user must configure the data source prior to executing the RunSQL.exe program. The value is typically a short name, possibly containing spaces. The DSN cannot be a "Local" DSN, and must be configured within the "System" tab of the Windows administrative tool.

username

Optional. This is the username for the ODBC Data Source defined above. Not all DSNs and databases require usernames. However, if the target database requires a username and password, both must precisely match the values configured in the Windows "Control Panel". The default username is "None".

password

Optional. This is the password for the ODBC Data Source defined above, if required. The value must precisely match the password configured in the Windows "Control Panel". The default username is "None".

usedb

This is the name of a database (within the main database) that prefixes all table names and column names. The value is not required, but can be used to shorten and simplify any queries. The value corresponds to the database segment name that is specified via an SQL "USE" clause. The default value is "None", indicating that no "USE" is implied.

maxresults

Optional. This is a positive integer value that indicates the maximum number of rows returned by a query. The value is used ONLY if the SQL statement generates output, and is otherwise not used. This value is important to keep a web page or program from becoming completely overrun with query results. (For example, it is not uncommon for a single query to access millions of results.) The default value is "50", indicating that a maximum of 50 rows are returned by any query.

sqlstmt

Optional. This is the SQL statement to execute automatically when the RunSQL.exe program is launched. The value can be omitted from the configuration file, in which case the SQL statement MUST be specified on the command line to the RunSQL.exe program. If the "sqlstmt" directive is present in the file then any command line query is ignored. (The directive in the file overrides any command line option, for security.) Note that this directive can be followed any valid SQL statement, including "INSERT", "SELECT", "DELETE", and others, to a maximum of 500 characters.

listfmt

Optional. This value is used ONLY if the SQL statement generates output, and otherwise is not used. The value is one of the following: "html", "xml", or "text". The value affects the output-listing format: "html" will display the results in raw HTML format; "xml" will display the results as an XML document; "text" will display the results as a list of configuration items. These formats are described in a later section of this Appendix.

As shown above, the only required parameter in the file is the "dsnname". If any of the other directives are omitted from the configuration file, then the default values explained above are used.

The "listfmt" Configuration File Directive

The "listfmt" configuration file directive describes the format of the results query, and provides three separate options, depending upon the requirements of the user.

- **The "text" List Format.** This format displays a listing of name and value pairs, in the Sigma configuration file format discussed above. Each configuration item name is given the column and row name, so each name is unique. The values are automatically truncated at 500 characters. The resulting format is well suited for creating configuration files for use with the Sigma "Macro" facility.

- **The "xml" List Format.** This format displays the results as an XML document, suitable for use with AJAX programming, or other XML applications. The resulting list is a well-formed XML document.

- **The "html" list Format.** This format displays the results as a raw HTML listing, with <TR> and <TD> tabs to create the contents of an HTML table. The <TABLE> anchors are omitted from the listing. This type of list is especially well suited for Sigma Macro execution using the "@@!" type macro discussed earlier.

If the "listfmt" directive is omitted from the RunSQL.exe configuration file, or if the value for the "listfmt" directive is other than "html", "xml", or "text", then the program uses the "text" format for query results.

Error Handling

The RunSQL.exe command handles both internal errors, and also relays SQL and ODBC errors to the user. The particular error message format depends upon the "listfmt" configuration file directive displayed above. If the user has specified "text" then the error message is displayed to standard output as a text message; if the user has specified "xml" then the error message is a small XML document; in the case of an HTML document, the text message is formatted as a single row of a table.

Error messages are easily parsed by a user program. The easiest way to see an error message is to specify an unknown DSN name, or some other problem with the configuration file. Based upon the particular "listfmt" setting, the user can easily determine through inspection the best way to handle the error.

Creating HTML Documents From SQL Queries

The simplest way to create an HTML document that contains the results of a query is to use the "@@!" Sigma Macro, documented in Section 4 of this manual. (This type of macro runs an external command, and then displays the standard output of that program as part of the HTML document.)

For example, if the configuration file shown earlier in this Appendix is copied to the "SQL\mytable.cnf" file of the Sigma root directory, the following HTML document will execute the SQL query, and display the contents in the end-user's web browser:

```
<html>
<head><title>SQL Query Results</title></head>
<body>
<center>
<h4>SQL Query Results</h4>
<table width=100% border=1>
@@!RunSQL.exe ../SQL/mytable.cnf@@
</table>
</center>
</body>
</html>
```

The above text can be copied to the file "sigma-web\050_Query.html" in the Sigma root directory, to create a "Query" tab on the browser. When the user clicks the tab, the HTML document is accessed, the "@@!RunSQL.exe@@" macro is replaced by the query results, based upon the query found in the "mytable.cnf" file.

Note that the full pathname to the RunSQL.exe program is not required, since this command resides in the "system" directory of the Sigma root directory (and this directory is always incorporated into the HTTP server's search path.) Further note that the pathname to the "mytable.cnf" file must contain a "../" reference, since the working directory of the HTTP server is the "cgi" directory.

Using CSS Styles with SQL HTML Output

The above example is all that is necessary to actually create a usable HTML document. However, except for the style attributes of the <TABLE> tag (such as "width" and "border"), the format of the table will be vanilla HTML defaults. The user may wish to modify this to make the output report more conformal to other screens.

The look of the HTML table, containing the SQL results, is easily modified via CSS styles. The programmer merely references the "class" attributes of the table, which are automatically added to the HTML output. This permits users to modify table colors, column widths, and other characteristics of the display output. The class names for the HTML elements are readily available by inspecting the HTML source.

Note that this will entail modifying the "styles.css" file, as described in Section 7 of this manual (since the HTML <HEAD> tag is contained in the s-header.html file, and cannot be overridden by the HTML document described here.) Since the class names for the various HTML elements are very unique, there should be no issue with adding extra styles for the various SQL HTML output tables, simply by appending the styles to the "styles.css" file of the system.

Conclusion

Various other techniques can be used to interface SQL to Sigma. For example, a perfectly suitable way of updating HTML files with database query results is to specify a "listfmt" of "text", and then creating a batch file that creates the ".cnf" files within the "sigma-web" directory, as discussed in Section 3 of this manual. The batch files can be scheduled on an hourly, daily, or weekly basis using the "Scheduler" function of Sigma. This is an especially useful technique if there is some database value that takes a long time to derive, such as counting up distinct values spanning many different rows.

To debug the system, an administrator can use the "System > ODBC" Sigma screen, documented in Section 3 of this manual, to test queries. This screen can be used to perfect and refine SQL statements interactively. The RunSQL command can also be executed at a command prompt to immediately see the output of the SQL statements, including any error output.

As with many of the Sigma facilities, the exact nature of the application will dictate the form of the usage. Ample information on standard SQL can be found in a variety of locations, including the web. Contact the vendor for specific assistance.

Appendix C: Web.cnf Config File

This appendix provides a description of the Sigma Web.cnf file directives. This file exists in the "/s-cgi" directory, and provides specific directives that permit the user to adapt the look and feel of the program. The file is created on startup (if it does not exist.)

Many of the values in the file are overridden by other parts of the program, hence are useful only if the "web.exe" program is applied to some new framework. The values should generally not be modified without careful consideration, and consultation with vendor engineering. The following "web.cnf" directives are supported:

COLOR.BG_ON

This directive provides the default color of active tabs. The default value of 000000 (black) is used if no other color information is available. This value is usually overridden by other elements of the program.

COLOR.BG_OFF

This directive provides the default color of non-active tabs. The default value of EEEEEE (gray) is used if no other color information is available. This value is usually overridden by other elements of the program.

COLOR.BORDER

This directive provides the default color of border frames. The default value of 999999 (dark gray) is used if no other color information is available. This value is usually overridden by other elements of the program.

COLOR.TEXT_ON

This directive provides the default color of text used in active tabs. The default value of FFFFFF (white) is used if no other color information is available. The value is usually overridden by other elements of the program.

COLOR.TEXT_OFF

This directive provides the default color of text used in non-active tabs. The default value of 555555 (dark gray) is used if no other color information is available. The value is usually overridden by other elements of the program.

FONT.FACE

This directive provides the default font face used in the program. The default value of "Helvetica" is used if no other font information is available. The value is usually overridden by other elements of the program.

FONT.SIZE

This directive provides the default font size used in the program. The default value of 2 is used if no other font size information is available. The value is usually overridden by other elements of the program.

PATH.CGI_EXEC

This directive specifies the URL pathname used to find the main program, by default "/s-cgi/web.exe" This value has multiple arcane purposes, such as to redirect frames and borders when applying the program within an IFRAME, or some other framed system. The value should generally not be modified without consulting support and professional services.

PATH.START_URL

This directive specifies the URL pathname used to find the tabbed programs and tab structure, by default the pathname "/sigma-web/". (A trailing slash is required for this value.) This value permits alternate systems of navigation tabs to be specified. The value should generally not be modified without consulting support and professional services.

PATH.HTML_URL

This directive specifies the URL pathname used to find general HTML documents, by default the pathname "/s-html/:. (A trailing slash is required for this value.) This value permits alternate systems of HTML documents to be specified. The value should generally not be modified without consulting support and professional services.

PATH.FOOTER

This directive specifies the disk pathname used to find the footer file, displayed at the bottom of screens. The default path "/s-html/s-footer.html" can be modified to point to some other footer value. The footer file is usually an HTML snippet that includes finalizing HTML information, including the closing </html> tag.

PATH.HEADER

This directive specifies the disk pathname used to find the header file, displayed at the top of screens. The default path "/s-html/s-header.html" can be modified to point to some other header value. The header file is usually an HTML snippet that includes initial HTML, a banner, and other markup information, including the starting <html> tag.

PATH.START_DIR

This directive specifies the disk pathname used to find tab programs, by default the path "../sigma-web". This value usually (but not necessarily) agrees with the "PATH.START_URL" value. (The "START_DIR" value specifies a disk path, whereas the "START_URL" value specifies a URL path.)

PATH.HTML_DIR

This directive specifies the disk pathname used to find html, by default the path "../s-html". This value usually (but not necessarily) agrees with the "PATH.HTML_URL" value. (The "HTML_DIR" value specifies a disk path, whereas the "HTML_URL" value specifies a URL path.)

PATH.SIDE_BAR

This directive, if included in the web.cnf file, specifies the disk pathname used to find an HTML file that is used with a side bar. The directive can be used to change the entire navigation scheme of the program, so that a side bar is used instead of nested tabs. The value should point to an existing HTML file snippet (without <html> or </html> directives) that is displayed to the left of the main window. See more information below. If the directive is specified, its value must point to a valid HTML file on the system.

SIZE.BUTTON

This directive specifies the minimum width of buttons on the system. The default value of "100" is used if no other sizing information is available. The value is usually overridden by other elements of the program.

SIZE.HTML_PAD

This directive specifies the size of an arbitrary textual pad that is placed between the tabs and the main HTML, used to render the tabs quickly on some browsers. The default value of 5000 characters generally affects

only Mozilla type browsers, and is useful for rendering the tabs in advance of other data. (This is a characteristic of Firefox and a few other browsers, placed here only to modify the aesthetics of the browser rendering engine, and otherwise having no purpose.)

PATH.SIDE_BAR – Creating a Navigation Side Bar

The user can significantly alter the navigation scheme used by the Sigma Framework, replacing the default "tabbed" navigation style with a side bar that appears to the left of the system. This is accomplished by setting the "PATH.SIDE_BAR" directive to point to an HTML file containing markup code and links of an arbitrary nature.

If the "PATH.SIDE_BAR" directive exists, and if the value points to the pathname of an accessible HTML file, then the navigation tabs are disabled and the HTML file provides links to the program. (These link URLS can simply be cut and pasted from a browser address bar.) The PATH.SIDE_BAR filename can contain any arbitrary code, including tables to set the width of the side bar, miscellaneous graphics, and other markup.

A simple navigation bar is shown below. The operator can cut and paste the following simple markup code to a file such as "s-html/navigation_bar.html", and then configure the PATH.NAV_BAR directive of the "web.cnf" file to reference this file, such as the following directive:

```
PATH.SIDE_BAR ../s-html/navigation_bar.html
```

The presence of both the above directive and the HTML markup file below will result in navigation tabs being removed from the system, and a simple sidebar being created that permits the user to drill down into the "Home" screen, the "Dashboard" screen, and the "Reports" screen.

This provides a simple technique to customize the navigation so that a more traditional side-bar approach is used.

```
<!-- Example Navigation Bar -->

<center>

<!-- This links to the "site_map.html" file. -->

<p>
<a href=/s-cgi/web.exe?Util/Site_Map.exe>Site Map...</a>
<p>

<!-- Start navigation table. -->

<table width=150 cellpadding=5 cellspacing=5>

<tr><td bgcolor=555555>
<a href=/>
<font color=ffffff>
Home
<font></a> </td>
</font>
</tr>

<tr><td bgcolor=555555>
<a href=/s-cgi/web.exe?150_Dashboards.exe>
<p>
<font color=ffffff>
Dashboards
</font></a></td>
</tr>

<tr><td bgcolor=555555>
<p>
<a href=/s-cgi/web.exe?400_Reports.dir/100_Excel.exe>
<font color=ffffff>
Reports
<font></a></td>
</tr>

<!-- Add additional links here. -->

</table>
</center>
```

For Additional Help And Information…

Detailed specifications regarding the CorreLog Server, add-on components, and resources are available from our corporate website. Test software may be downloaded for immediate evaluation. Additionally, CorreLog is pleased to support proof-of-concepts, and provide technology proposals and demonstrations on request.

CorreLog, Inc., a privately held corporation, has produced software and framework components used successfully by hundreds of government and private operations worldwide. We deliver security information and event management (SIEM) software, combined with deep correlation functions, and advanced security solutions. CorreLog markets its solutions directly and through partners.

We are committed to advancing and redefining the state-of-art of system management, using open and standards-based protocols and methods. Visit our website today for more information.

CorreLog, Inc.
http://www.CorreLog.com
mailto:support@CorreLog.com

Alphabetical Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W