eXtropia WebDB Guide

Version 2.0 Nov, 27 2001

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

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

(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 WebDB

1.1 Overview

Welcome to WebDB, eXtropia's Web-based Database Management software. WebDB allows you to quickly and easily set up simple databases for the web. It's simple model lends itself well to 90% of databases that are put on the web without customization of the Perl code. This flexibility is demonstrated in this model as we provide sample **news application**, **project tracker**, **inventory manager**, and **address book** as sample applications based on the WebDB template.

In fact, WebDB is a bit different from most other eXtropia applications. There really is not such thing as a *webdb.cgi*. Frequently, you, the user, will decide what type of database WebDB should be (eg an address-book) and then modify the configuration parameters in webdb.cgi to fit your requirements. At this point, you would rename *webdb.cgi* to reflect your application (eg *address_book.cgi*).

> **NOTE: *webdb.cgi* consists solely of configuration parameters. It contains *NO* code logic. Logic is contained in a module called *WebDB.pm* which we discuss later. Thus, *webdb.cgi* is actually a configuration file that doubles as the executable the web server uses to launch the application. Changing the settings and renaming it to something like "address_book.cgi" does not hinder us from providing you with application updates to the core *WebDB.pm* module.**

This guide is meant to walk you through installing, configuring, and customizing WebDB. Most of you will be able to unpack the application and run it on your server after setting the appropriate file permissions.

In addition, since we recognize that all the eXtropia WebWare2 applications follow a similar design, we have also heavily referenced an extremely detailed eXtropia Applications Guide which we describe in the *Further References* chapter.

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 documentation.

1.2.1 Where to Find The Credits...

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!

1.2.2 How To Get Credit: Documentation

The best way to get your name in there is to offer us suggestions to improve this documentation. Also, we do not intend this to be the only documentation for WebDB. We realize that different people have different ideas of how to approach learning software.

To that end, we would welcome links to any alternative documentation including documentation that may be translations of this or other guides to other languages. We love supporting this stuff *globally* and we love to see alternate language support grow. Over the past six years the eXtropia team has been located in USA, Europe, and as you read this, in Asia's information hub, Singapore.

1.2.3 How To Get Credit: Programming/Community

As with other open source communities, we also welcome additions, suggestions, and cool hacks (http://www.extropia.com/hacks/) to the software originated at eXtropia. The more we incorporate from you, the better our software will be.

1.3 How To Read This Guide

At first glance, you might think there is a lot to this guide. The PDF version is over 60 pages long. However, we feel that it is currently organized in a way that the most people will benefit.

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, we also realize that you have better things to do than read documentation, so our organization of this guide reflects this.

If you are already familiar with eXtropia applications, you should be able to skim through the the chapter on installation and set up the application pretty quickly. If you are completely new at the process, we suggest that you also pick up a copy of the eXtropia Applications Guide described in *Appendix A: Further References*.

Chances are good that you may actually fall into more than one of these or in-between, so we'll leave it up to your best judgement about how you should read the guide. To help you decide which sections would be most useful to read, we have abandoned our practice of distributing huge read me text files and instead have fully hyperlinked HTML pages and PDF files which include hyperlinked table of contents. Enjoy!

1.4 What's Included in WebDB

As we stated before, WebDB is not is not an application by itself. Rather it is an *enabler* that allows you to create applications with it. A WebDB application is nothing without a database behind it; your database is your application!

To demonstrate this, we have distributed four sample configurations of WebDB contained in *news.cgi*, *project_tracker.cgi*, *inventory_manager.cgi*, and *address_book.cgi*. Note that eXtropia applications are **Object-Powered**. This means that the WebDB logic is not inside of the CGI script. The CGI script merely sets up configuration variables unique to your environment and requirements and then subsequently calls the *WebDB object* to configure and run it.

The following describes these sample applications in greater depth:

- **address_book.cgi**

  The Address Book can be used to track friends, family, as well as business contacts. Nearly every organization with an Intranet has an on-line internet address book.

- **project_tracker.cgi**

  The Project Tracker allows you to track projects, their status, and the various relationships that effect a projects delivery. Some additional features are enabled such as the capability to color projects based on their status. An overdue project can go into the red zone!

- **news.cgi**

  The News application allows you to post and read news articles. Useful for a web site to manage the news items that it displays. It can also be used as an extremely simple groupware tool where a sales force can occasionally update news of what is going on in their areas. There is an administrative script as well.

- **inventory_manager.cgi**

  The Inventory manager is useful for managing the contents of a web store database. You can also use it to track collections such as CD collections and whatever else you might be interested in.

- **recruit.cgi and recruit_admin.cgi**

- **guestbook.cgi**

- **bbs.cgi**

  There are many uses for a bulletin board system on the Web. A ‘‘BBS’’ can be used to make information continually available that would normally be shared only during infrequent meetings. Within a BBS, users can discuss topics, reading the responses to their ideas at their leisure. On the Internet, vendors can use a BBS to discuss their products or offer technical support. Additionally, users on the Internet can set up a BBS spontaneously to discuss things that interest them or just for fun. The main feature of WebBBS is that it allows a user to post messages as well as post replies to existing messages. WebBBS keeps track of which messages are posts and which ones are replies and displays them in a hierarchical tree-like fashion.

- **bug_tracker.cgi**

  BugTracker is a useful tool for programmers of all types and experience levels. Make sure your projects stay on track by keeping track of bugs and feature requests. Many clients now require a listing of such bugs. BugTracker is a must for all software projects.

- **comment_form.cgi**

  Allows a user to enter comments and send it via email to the administrator. The comments are also saved to the administrator's choice of database.

- **doc_manager.cgi**

  Document Manager is ideal for tracking and using documents when on the go. Useful for coordinating files among a decentralized team.

- **expense.cgi and expense_admin.cgi**

  A great tool for the office manager to help control expenses while maintaining an easy to use listing of all expense requests. Permits approval and data tallying.

- **jump_form.cgi**

  A simple script that takes a listing of URLs and permits the user to be redirected with a click of a button.

- **mlm.cgi (Mailing List Manager)**

  Allows list users to add and remove themselves as well as enter in posts for approval.

- **survey.cgi and survey_admin.cgi**

  Ever want to find out what your users are interested in? Here's your chance! Allows easy customization to ask questions of all sorts while emailing it to the administrator and maintaining a database of the information.

- **tell_a_friend.cgi**

  Allow users to send email to a friend with a preset link based on where the user came from. Can also keep track of who is doing what.

- **todo.cgi**

  A To Do List! Enter in your tasks to help manage your day and projects.

- **webcal.cgi**

  A very powerful groupware calendar system designed for a network of professionals over a wide area network. Permits user authentication, Day, Month, Year views, event spanning, and recurring events.

Enjoy!

The eXtropia Team

;o)

2 Installation Guide

2.1 Overview

If you just want to install the application and start running it, you’ve come to the right place!

This chapter specifically centers around the steps involved with installing WebDB. However, as mentioned in the previous chapter, we do assume quite a bit of knowledge of both general CGI and how eXtropia applications work. However, we liberally refer back to the general eXtropia Applications Guide described in *Appendix A: Further References*.

2.2 The 12 Step Checklist

We start out first with our famous 12-Step checklist to installing any eXtropia application. For further info on the history and details behind the 12-Step list, consult the eXtropia Applications Guide.

As a quick reminder, here are the 12-Steps:

1. **Prepare your environment to run CGI applications.**
2. **Obtain the application installation file (webdb.tar).**
3. **Unpack the application installation file on your web server.**
4. **Set the file and directory permissions correctly.**
5. **Fix the reference to the Perl executable on first line of the application executable (webdb.cgi).**
6. **Define site-specific parameters by modifying configuration variables in the application executable (webdb.cgi).**
7. **Modify the look-and-feel by editing or creating views (webdb/Views/eXtropia).**
8. **Run the application using a web browser.**
9. **Debug the application if any debugging is required.**
10. **Review the security of the application.**

    In particular, move administrative and data files out of web documents tree.
11. **Submit the application for user testing.**
12. **Register yourself as a user by sending an email to register@extropia.com.**

2.3 Step 1: Prepare your Site

The first step in the process of installing the application is to prepare your environment for CGI execution. If you are not sure how to do this, consult the eXtropia Applications Guide.

2.4 Step 2: Obtain the Installation File

To obtain the latest version of this application, we recommend that you go straight to http://www.extropia.com/. Mirror-sites and CD-ROMS are not guaranteed to have the truly latest versions of our software at any given time.

2.5 Steps 3 and 4: Unpacking the Archive File and Setting Permissions

Though the details of unpacking and permissioning are covered in depth in the eXtropia Applications Guide discussed in *Appendix A: Further References*, we include the following basic guidelines for your convenience:

2.5.1 UNIX-based web servers

Use *tar xvfz webdb.tar.gz* to unpack.

If that does not work,

Use *gunzip webdb.tar.gz*

THEN

Use *tar xvf webdb.tar* to unpack.

2.5.2 Windows-based web servers or Windows-based workstation

Use a program such as *WINZIP* to unpack.

2.5.3 Mac-based web server

Use a program such as *Stuffit Expander*.

Once unpacked, the application files will expand into the default directory structure shown in the directory tree further below. The following list provides a set of rules for permissioning:

- **The application executables (eg. address_book.cgi) should have read and execute permission (r-xr-xr-x).**

- **The Datafiles directory and all sub-directories of the Datafiles directory, should have read, write and execute permissions (rwxrwxrwx).**
- **Files within the Datafiles directory and all files within sub-directories of the Datafiles directory, should have read and write permission (rw-rw-rw).**
- **All modules, views, and the application object (WebDB.pm) should have read-only permission (r--r--r--).**
- **The Modules directory and all its subdirectories as well as the View directory and all its subdirectories should have read and execute privileges (r-xr-x r-x).**

Directory structure and permissioning:

```
    webdb                                         r-xr-xr-x
        address_book.cgi                          r-xr-xr-x
        inventory_manager.cgi                     r-xr-xr-x
        news.cgi                                  r-xr-xr-x
        project_tracker.cgi                       r-xr-xr-x
        Views                                     r-xr-xr-x
            eXtropia
                AuthManager                       r-xr-xr-x
                    CGIViews.pm                   r--r--r--
                StandardTemplates                 r-xr-xr-x
                    BottomFrameView               r--r--r--
                    ErrorDisplayView              r--r--r--
                    FrameView                     r--r--r--
                    TopFrameView                  r--r--r--
                    PageTopView                   r--r--r--
                    PageBottomView                r--r--r--
            WebDB                                 r-xr-xr-x
                AddressBook                       r-xr-xr-x
                    AddEventEmailView.pm          r--r--r--
                    ModifyEventEmailView.pm       r--r--r--
                    DeleteEventEmailView.pm       r--r--r--
                    DetailsView.pm                r--r--r--
                    AddRecordView.pm              r--r-r--
                    BasicDataView.pm              r--r--r--
                    ModifyRecordView.pm           r--r--r--
                    RecordSetDetailsView.pm       r--r--r--
                    SearchBoxView.pm              r--r--r--
                InventoryManager (same views as AddressBook)
                News (same views as AddressBook)
                ProjectTracker (same views as AddressBook)
        Apps                                      r-xr-xr-x
            eXtropia                              r-xr-xr-x
                WebDB.pm                          r--r--r--
        Modules                                   r-xr-xr-x
            There are a heck of a lot of modules.
            You can see a list of commonly used
            modules in the eXtropia Application Guide.
        Datafiles                                 rwxrwxrwx
            address_book.dat                      rw-rw-rw-
            address_book_users.dat                rw-rw-rw-
            address_book.log                      rw-rw-rw-
            [each of the other three apps have these three files]
```

As you can see above, unlike most other eXtropia applications that have only one application executable, WebDB has four. This is because we wanted to give you a better sense of the possibilities by example. However, don’t let the number of application executable unnerve you.

If you read them, you will see that they are practically the same. For the most part, the only differences are in the views, the data source configuration, and the configuration of the view parameters.

2.6 Step 5: Modify the Perl Path

In order to run the application you must make sure that the first line of the application executable (eg. *address_book.cgi*) points to your local version of Perl. If you open the application executable in a text editor, you will see that by default, the application expects that Perl is installed in the directory */usr/bin*. That is, by default, the first line of the file reads:

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

If your copy of Perl is located elsewhere, you must modify this one line to reflect the difference. For example, if your copy of Perl is located in the directory */opt/bin*, the first line will read:

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

If you have questions about how to find out where your copy of Perl is located, which version of Perl is required, or if you are using a Windows-based web server, consult the eXtropia Applications Guide discussed in *Appendix A: Further References*. Also remember that the reference to Perl must absolutely be on the very first line of the file.

2.7 Step 6: Configure the Application

NOTE: If you are one of the lucky ones, and you are using a web server configuration that matches the one we use at eXtropia, you can probably fire up the application at this point and use it in its default state.

In order to find out, we recommend skipping to Step 8 right now and trying to run the application using a web browser.

Most likely, you will still need to return to this section to make the application work exactly the way you want it to rather than the way it is configured by default. But at this stage it is useful to see if it works out-of-the-box.

When configuring this application, you will be concerned primarily with modifying the configuration parameters defined in the application executable. We have already discussed how the configuration of an eXtropia application executable works in Chapter 3 so we won’t repeat that here.

Instead, we will look at each of the configuration parameters in the application executable and explain what they do relative to this specific application. But as we said earlier, all the sample application executables have the same basic configuration. If you would like more help with configuration, you can also read the Configuration by Example section that appears later in this chapter.

However, it is crucial that you are very comfortable with the details presented in Step 6 of the *Installation* chapter in the eXtropia Applications Guide discussed in *Appendix A: Further References*.

In particular, you should understand:

- **How configuration works.**
- **How to modify configuration parameters.**
- **How to debug configuration modification errors.**

2.7.1 The Configuration Preamble

Like all eXtropia applications, the application executable (eg *address_book.cgi*) begins with a preamble. The details of the preamble are explained in Chapter 3 so we won't go over them here. Of course, 99.9% of the time, you will never need to modify this part anyway. So dont get too nervous if the code looks unfamiliar.

Following the preamble are the configurations for the ten major components used by this application.

- **Session**
- **Session Management**
- **Authentication**
- **Authentication Manager**
- **DataHandlerManager**
- **DataSource**
- **Mailing**
- **Logging**
- **View**
- **Filter**

The following figure shows the configuration hierarchy for the application:

[IMAGE]

Let's take a look at each type of configuration individually.

2.7.2 Session and Session Manager Setup

Sessions give the application the ability to remember what has already happened in an application workflow and are necessary because HTTP is stateless. A session manager is a tool that allows you to more easily manage a session.

You can access a detailed discussion of session and session managers in the eXtropia Applications Guide discussed in *Appendex A: Further References*.

In this application however, they are primarily used to maintain authentication information. The following example shows how the sessions are configured by default.

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

my @SESSION_MANAGER_PARAMS = (
    -TYPE           => 'FormVar',
    -CGI_OBJECT     => $CGI,
    -SESSION_PARAMS => \@SESSION_PARAMS
);
my $SESSION_MGR = Extropia::SessionManager->create(@SESSION_MANAGER_PARAMS);
my $SESSION     = $SESSION_MGR->createSession();
```

2.7.3 Authentication Setup

Authentication allows an application to check a user against a list of valid users in a data source and decide whether or not any given user should have access to perform some function. If you are interested, you can read more about Authentication in the eXtropia Applications Guide discussed in *Appendix A: Further References*.

The following shows an example of how the authentication variables are configured by default in WebDB:

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

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

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

my @AUTH_PARAMS = (
    -TYPE                                => 'DataSource',
    -USER_DATASOURCE_PARAMS              => \@AUTH_USER_DATASOURCE_PARAMS,
    -ENCRYPT_PARAMS                      => \@AUTH_ENCRYPT_PARAMS,
    -AUTH_CACHE_PARAMS                   => \@AUTH_CACHE_PARAMS,
    -ADD_REGISTRATION_TO_USER_DATASOURCE => 1,
    -USER_FIELDS_TO_DATASOURCE_MAPPING   => \%USER_FIELDS_TO_DATASOURCE_MAPPING
);
```

2.7.4 Authentication Manager Setup.

Generally an application never really talks to an authentication object directly. Usually, it talks to an authentication manager. The authentication manager performs all the visible features of authentication such as presenting a registration or login screen to allow the user to register or log on to the application. Authentication managers are discussed further in the eXtropia Applications Guide.

The following example shows an example of how the authentication manager is configured in WebDB by default:

```
my @AUTH_REGISTRATION_DH_MANAGER_PARAMS = (
    -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 => [
                'auth_username',
                'auth_firstname',
                'auth_lastname',
                'auth_email'
    ],
    -IS_EMAIL => [
                'auth_email'
    ],
    -UNTAINT_EMAIL => [
                '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_PARAMS = (
    -TYPE                => 'CGI',
    -SESSION_OBJECT      => $SESSION,
    -AUTH_VIEWS          => './Apps/Extropia/MLMViews/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_PARAMS      =>
            \@AUTH_REGISTRATION_DH_MANAGER_PARAMS
);
```

2.7.5 Data Handler Manager Setup

As we have gone over in the eXtropia Applications Guide, data handlers perform a number of data filtering and validation functions on incoming form data. Data handler managers manage multiple types of data handlers in a way that makes it simple to perform many handling functions within an application in only a few lines of code.

In this application there are two forms that must be handled:

- **The Add New Entry Form**

  This form allows users to add records. This view is defined in the file *Views/WebDB/AppName/AddRecordView.pm*.

- **The Modify Entry Form**

  This form provides a simple interface through which to modify records. This view is defined in the file *Views/WebDB/AppName/ModifyRecordView.pm*.

An example of how WebDb (specifically *addressbook.cgi*) is configured by default for HTML form validation is shown below:

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

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

Note that the fields used in the data handler manager configuration relate specifically to the NAME values on the forms. Thus, the data handlers will expect to have a user-supplied value coming from HTML form elements such as:

```
    <INPUT TYPE = "TEXT" NAME = "email">
```

Notice that the `email` NAME corresponds to the name used in the `-IS_EMAIL`, `-IS_FILLED_IN`, and `-ESCAPE_HTML_TAGS` data handler rules.

2.7.6 Data Source Setup

As you might imagine, the core feature of the application is the data source itself. The data source contains all of the data being managed. As such, we use an *Extropia::DataSource*. The setup of this module is discussed in further detail in the eXtropia Applications Guide.

The following is an example of how the data source is configured by default in *news.cgi* (one of the apps that comes with the WebDB package).

```
my @DATASOURCE_FIELD_NAMES = qw(
    record_id
    subject
    abstract
    full_text
    username_of_poster
    group_of_poster
);

my $INPUT_WIDGET_DEFINITIONS = {
    subject => [
        -DISPLAY_NAME => 'Subject',
        -TYPE         => 'textfield',
        -NAME         => 'subject',
        -DEFAULT      => '',
        -SIZE         => 30,
        -MAXLENGTH    => 80
    ],
    abstract => [
        -DISPLAY_NAME => 'Abstract',
        -TYPE         => 'textfield',
        -NAME         => 'abstract',
        -DEFAULT      => '',
        -SIZE         => 30,
        -MAXLENGTH    => 80
    ],
    full_text => [
        -DISPLAY_NAME => 'Full Text',
        -TYPE         => 'textarea',
        -NAME         => 'full_text',
        -DEFAULT      => '',
        -ROWS         => 6,
        -COLS         => 30
    ]
};

my @INPUT_WIDGET_DISPLAY_ORDER = qw(
    subject
    abstract
    full_text
);

my @DATASOURCE_CONFIG_PARAMS = (
    -TYPE            => 'File',
    -FILE            => './Datafiles/webdb.dat',
    -FIELD_DELIMITER => '|',
    -FIELD_NAMES     => \@DATASOURCE_FIELD_NAMES,
    -FIELD_TYPES     => {
        item_id => 'auto'
    },
);
```

2.7.7 Mail Setup

One of the most important features of the application is its ability to send email event notifications. Specifically, the application sends three types of emails:

- **Add Event Notifications**

  The application administrator may receive a notification email that includes the details of the addition.

- **Delete Event Notifications**

  The application administrator may receive a notification email that includes the details of the deletion.

- **Modify Event Notification**

  The application administrator may receive a notification email that includes the details of the modification.

Thus, you are going to have to configure an *Extropia::Mail* driver to perform the mailings. Mail driver configuration is discussed in further detail in the eXtropia Applications Guide. The following example shows how mail is configured on WebDB by default.

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

    my @EMAIL_DISPLAY_FIELDS = qw(
        category
        fname
        lname
        phone
        email
        comments
    );

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

    my @ADD_EVENT_MAIL_SEND_PARAMS = (
        -FROM     => 'you@yourdomain.com',
        -TO       => 'you@yourdomain.com',
        -SUBJECT  => 'WebDB Addition'
    );

    my @MODIFY_EVENT_MAIL_SEND_PARAMS = (
        -FROM     => 'you@yourdomain.com',
        -TO       => 'you@yourdomain.com',
        -SUBJECT  => 'WebDB Modification'
    );
```

WARNING: If you have completed the mail configuration but your application is not mailing, you might check to make sure you have enabled mailing in the `@APPLICATION_SETUP` array. Remember that you must set `-SEND_EMAIL_ON_ADD_FLAG`, `-SEND_EMAIL_ON_DELETE_FLAG`, and `-SEND_EMAIL_ON_MODIFY_FLAG` to 1 if you want the application to actually mail the user or administrator respectively.

NOTE: If you are wondering how to change the body of the email, check out the *AddEventEmailView.pm*, *ModifyEventEmailView.pm*, and *DeleteEventEmailView.pm* files in the *webdb/Views/Extropia/WebDB* directory as well as the Look-and-Feel section later in this chapter and the Look-and-Feel chapter of the *eXtropia Application Guide*.

2.7.8 Log Setup

It is possible that you will also wish to maintain a log of errors, accesses and miscellaneous actions. Because of this, the application uses *Extropia::Log*. The configuration of *Extropia::Log* is discussed in further detail in the eXtropia Applications Guide.

```
    my @LOG_CONFIG_PARAMS = (
        -TYPE             => 'File',
        -LOG_FILE         => './Datafiles/webdb.log',
        -LOG_ENTRY_SUFFIX => '|' . _generateEnvVarsString() . '|',
        -LOG_ENTRY_PREFIX => 'WebDB|'
    );
    sub _generateEnvVarsString {
        my @env_values;

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

Note that the `_generateEnvVarsString()` method is also discussed in the eXtropia Applications Guide under the section discussing the Log configuration.

2.7.9 View Setup

Views define the application's user interface. You can find all the views for this application in the *webdb/Views/eXtropia* directory. For example, if you want to modify the way the *Add Record* form looks for the News example application, you would edit the file *webdb/Views/Extropia/WebDB/News/AddRecordView.pm*.

The following is a list of the views and their descriptions for WebDB in the webdb/Views/Extropia/WebDB/AppName:

- **AddRecordView.pm**

  Defines the user interface for the *Add Record* form.

- **BasicDataView**

  Defines the user interface for the main application frame in which searches are displayed in using HTML tables.

- **ModifyRecordView.pm**

  Defines the user interface for the *Modify Record* form.

- **DetailsView.pm**

  Defines the user interface that presents the full record details

- **AddEventEmailView.pm**

Defines the contents of the email body sent after add events

- **ModifyEventEmailView.pm**

  Defines the contents of the email body sent after modify events

- **DeleteEventEmailView.pm**

  Defines the contents of the email body sent after delete events

- **RecordSetDetailsFooterView.pm**

  Defines the footer for the record set detail (eg an ending </TABLE> tag).

- **SearchBoxView.pm**

  Defines the user interface for the search box shown on the top of the BasicDataView.

The template views stored in *webdb/Views/Extropia/StandardTemplates* are listed below:

- **BottomFrameView.pm**

  Defines the bottom application frame.

- **ErrorDisplayView.pm**

  Defines the error reporting user interface.

- **FrameView.pm**

  Defines the base application frame.

- **TopFrameView.pm**

  Defines the top application frame.

- **PageTopView**

  Defines the page header

- **PageBottomView**

  Defines the page footer

The authentication manager views are stored in the *webdb/Views/Extropia/AuthManager* directory and are listed below (*CGIViews.pm*).

- **CGIViews.pm**

Defines all the views required for CGI-based authentication.

The default configuration for the views in WebDB, specifically *projectmanager.cgi*, are shown below:

```
my @VALID_VIEWS = qw(
    AddRecordView
    BasicDataView
    DetailsView
    BottomFrameView
    FrameView
    ModifyRecordView
    TopFrameView
);

my @ROW_COLOR_RULES = (
    {'status' => [qw(Requested 99CC99)]},
    {'status' => [qw(In-Process CC9999)]},
    {'status' => [qw(Delivered CC9999)]}
);

my @FIELD_COLOR_RULES = (
    {'project_size' => [qw(Large BLUE)]},
    {'project_size' => [qw(Small ORANGE)]}
);

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,
    -COLOR_FOR_EVEN_ROWS      => 'E5E5E5',
    -COLOR_FOR_ODD_ROWS       => 'FFFFFF',
    -APPLICATION_LOGO         => 'web_db_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',
    -DOCUMENT_ROOT_URL        => '/',
    -IMAGE_ROOT_URL           => '/images',
    -SCRIPT_DISPLAY_NAME      => 'Project Tracker',
    -SCRIPT_NAME              => 'project_tracker.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',
    -EMAIL_DISPLAY_FIELDS     => \@EMAIL_DISPLAY_FIELDS,
    -HOME_VIEW                => 'BasicDataView',
    -FIELD_NAME_MAPPINGS   => {
        status              => 'Status',
        project_name        => 'Project Name',
        project_size        => 'Project Size',
        estimated_man_hours => 'Est. Man Hours',
        developer_name      => 'Developer',
        client_name         => 'Client',
        comments            => 'Comments'
    },
    -DISPLAY_FIELDS        => [qw(
        status
        project_name
        project_size
        estimated_man_hours
        developer_name
        client_name
        comments
    )],
    -SELECTED_DISPLAY_FIELDS        => [qw(
        status
        project_name
        developer_name
        client_name
        comments
    )]
);
```

> **NOTE: There is no need to put email-related views in the @VALID_VIEWS list because the email-related views are not actually displayed to a web browser They are just used to help compose email bodies. Similarly, you need not list views that are only called by other views in the @VALID_VIEWS list because the views called by other views are considered safe since they may not be specified from the web browser.**

Notice in particular the use of the color configuration parameters. These two view parameters can be used to color any row or field that matches the search criteria. For example, in the code above, you can see that if any row has a status field equal to *Requested*, the row will be colored green (99CC99). Likewise, any row that has the keyword *Large* in the project_size field will be colored blue.

2.7.10 Filter Setup

Filters are used to modify the contents of views before they are sent to the web browser. In other words, filters are components that transform the output of a view based on a set of logic. For example, using an *SSI Include* filter allows you to execute *SSI Include* commands inside of your HTML view output. You can find out more about filters and how they are set up in the eXtropia Applications Guide.

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

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

2.7.11 Application Setup

Finally, we are ready to put all the configuration variables into the application object configuration! This parameter contains references to all the configuration parameters that we just discussed. It also defines several application flags and values. It is this mega-package of configuration parameters that is sent to the application object (WebDB.pm) to initialize the application at run-time.

Let’s look at the parameters in greater detail:

- **-ADD_EMAIL_BODY_VIEW**

  This optional parameter specifies the view used to generate the body of the email sent to the administrator in the case of an add event.

- **-ADD_EVENT_MAIL_SEND_PARAMS**

  This optional parameter specifies a set of mail parameters used to define the details of the emails sent on add events if `-SEND_EMAIL_ON_ADD_FLAG` has been set to *1*. By default, all mail parameters must define `-TO`, `-FROM`, `-SUBJECT`, and `-BODY`.

However, optionally, they may define `-CC`, `-BCC`, `-REPLY_TO`, and `-ATTACH`. Note that the actual body of the email sent is defined by the view defined by *AddEventEmailView.pm*.

- **-ADD_FORM_DH_MANAGER_CONFIG_PARAMS**

  This optional parameter contains a reference to *Extropia::DataHandler* configuration parameters. This configuration is used to handle the *Addition* form.

- **-ADD_FORM_VIEW_NAME**

  This required parameter specifies the name of the view used to generate the *Add* form. The value is used primarily in error handling functions in which the application needs to return the user to the last form they were on.

- **-ALLOW_ADDITIONS_FLAG**

  This optional parameter specifies whether or not to allow additions. If set to 0, the application will not allow users to add entries.

- **-ALLOW_DELETIONS_FLAG**

  This optional parameter specifies whether or not to allow deletions. If set to 0, the application will not allow users to delete entries.

- **-ALLOW_MODIFICATIONS_FLAG**

  This optional parameter specifies whether or not to allow modifications. If set to 0, the application will not allow users to modify entries.

- **-AUTH_MANAGER_CONFIG_PARAMS**

  This optional parameter contains a reference to *Extropia::AuthManager* configuration parameters. The AuthManager will control authentication within your application. Whenever the `authenticate()` method is called, the user will have to provide a valid username and password. However, because the application stores valid logon results in the session object, they must only do so once in the application lifecycle.

- **-BASIC_DATA_VIEW_NAME**

  This required parameter specifies the name of the view used to generate the *Basic Data* view.

- **-CGI_OBJECT**

  This required parameter contains a reference to a *CGI* object (instantiated from Lincoln Stein's CGI.pm library). The *CGI* object will contain all of the data supplied by the user. It is used in several locations including the add, modify, and delete routines as well as in several views that must implement stickiness.

- **-DATASOURCE_CONFIG_PARAMS**

  This required parameter contains a reference to *Extropia::DataSource* configuration parameters. By default, all eXtropia applications use a File-based data source for cross-platform reasons. However, you can easily use this parameter to specify any other eXtropia data source type.

- **-DEFAULT_MAX_RECORDS_PER_PAGE**

  This optional parameter specifies how many records to display per page. If it is not specified, it is set to 10 by default.

- **-DEFAULT_SORT_FIELD1**

  This optional parameter specifies a field within the data source to sort on when data is returned from a search. Note that the field name should correspond to one of the fields specified in @DATA-SOURCE_FIELD_NAMES. For example if the first name field was specified and a data source search returned three names, *Michael*, and *Anthony* and *Jeremy*, the rows would be sorted to be *Anthony*, *Jeremy*, and *Michael*.

- **-DEFAULT_SORT_FIELD2**

  This optional parameter specifies a secondary field within the data source to sort on when data is returned from a search. Note that the field name should correspond to one of the fields specified in @DATASOURCE_FIELD_NAMES.

  For example if the first name field were specified as `-SORT_FIELD1` and the last name field were specified as `-SORT_FIELD2`, and a data source search returned three names *Michael Chopek*, and *Michael Budash* and *Jeremy Horland*, the rows would be sorted to be *Jeremy Horland*, *Michael Budash*, and *Michael Chopek*. Notice that *Budash* comes before *Chopek* as last name is specified as a secondary sort order to the first name.

- **-DEFAULT_VIEW_NAME**

  This required parameter specifies what view the application should display. Note that because of how it is defined, this parameter may be overridden with a view parameter specified in the incoming HTTP stream. Views are explained in more detail in the eXtropia Applications Guide.

- **-DELETE_EMAIL_BODY_VIEW**

  This optional parameter specifies the view used to generate the body of the email sent to the administrator in the case of a delete event.

- **-DELETE_EVENT_MAIL_SEND_PARAMS**

  This optional parameter specifies a set of mail parameters used to define the details of the emails sent on delete events if `-SEND_EMAIL_ON_DELETE_FLAG` has been set to *1*. By default, all mail parameters must define `-TO`, `-FROM`, `-SUBJECT`, and `-BODY`. However, optionally, they may define `-CC`, `-BCC`, `-REPLY_TO`, and `-ATTACH`. Note that the actual body of the email sent is defined by the view defined by *DeleteEventEmailView.pm*.

- **-DELETE_FORM_DH_MANAGER_CONFIG_PARAMS**

  This optional parameter contains a reference to *Extropia::DataHandler* configuration parameters. This configuration is used to handle the *Delete* form.

- **-DELETE_FORM_VIEW_NAME**

  This required parameter specifies the name of the view used to generate the *Delete* form. The value is used primarily in error handling functions in which the application needs to return the user to the last form they were on.

- **-LOG_CONFIG_PARAMS**

  This optional parameter contains a reference to *Extropia::Log* configuration parameters. The log will record important events such as additions, modifications, deletions, as well as notable errors such as attempts to load invalid views.

- **-MAIL_CONFIG_PARAMS**

  This optional parameter contains a reference to *Extropia::Mail* configuration parameters. The parameter is optional because a data source administrator may not wish to implement any emailing workflow. To turn off emailing, you must set `-SEND_EMAIL_ON_DELETE_FLAG`, `-SEND_EMAIL_ON_DELETE_FLAG`, and `-SEND_EMAIL_ON_DELETE_FLAG` equal to 0.

- **-MODIFY_EMAIL_BODY_VIEW**

  This optional parameter specifies the view used to generate the body of the email sent to the administrator in the case of a modify event.

- **-MODIFY_EVENT_MAIL_SEND_PARAMS**

  This optional parameter specifies a set of mail parameters used to define the details of the emails sent on modify events if `-SEND_EMAIL_ON_MODIFY_FLAG` has been set to *1*. By default, all mail parameters must define `-TO`, `-FROM`, `-SUBJECT`, and `-BODY`. However, optionally, they may define `-CC`, `-BCC`, `-REPLY_TO`, and `-ATTACH`. Note that the actual body of the email sent is defined by the view defined by *ModifyEventEmailView.pm*.

- **-MODIFY_FORM_DH_MANAGER_CONFIG_PARAMS**

  This optional parameter contains a reference to *Extropia::DataHandler* configuration parameters. This configuration is used to handle the *Modification Form*.

- **-MODIFY_FORM_VIEW_NAME**

  This required parameter simply specifies the name of the view used to generate the *Modify Form*. The value is used primarily in error handling functions in which the application needs to return the user to the last form they were on.

- **-REQUIRE_AUTH_FOR_ADDING_FLAG**

  This optional parameter specifies whether or not users must authenticate in order to add to the database. If it is set to 1, users must authenticate. If set to 0, users can add without logging on.

- **-REQUIRE_AUTH_FOR_DELETING_FLAG**

  This optional parameter specifies whether or not users must authenticate in order to delete from the database. If it is set to 1, users must authenticate. If set to 0, users can delete without logging on.

- **-REQUIRE_AUTH_FOR_MODIFYING_FLAG**

  This optional parameter specifies whether or not users must authenticate in order to modify the database. If it is set to 1, users must authenticate. If set to 0, users can modify without logging on.

- **-REQUIRE_AUTH_FOR_SEARCHING_FLAG**

  This optional parameter specifies whether or not users must authenticate in order to search the database. If it is set to 1, users must authenticate. If set to 0, users can search without logging on.

- **-REQUIRE_MATCHING_USERNAME_FOR_MODIFICATIONS_FLAG**

  This optional parameter specifies whether the user who originally added a record will be the only user allowed to modify the record. It is is set to 1, only the user may modify the record.

- **-REQUIRE_MATCHING_USERNAME_FOR_MODIFICATIONS_FLAG**

  This optional parameter specifies whether the user who originally added a record will be the only user allowed to delete the record. It is is set to 1, only the user who created the entry may delete the record.

- **-SEND_EMAIL_ON_ADD_FLAG**

  This optional parameter determines whether or not the application should send an email when it performs a successful addition. If it is set to 0, no emails will be sent. If it is set to 1, and you have properly defined `-MAIL_CONFIG_PARAMS` and `-ADD_EVENT_MAIL_SEND_PARAMS`, the application will send mail. By default, the parameter is set to 0 .

- **-SEND_EMAIL_ON_DELETE_FLAG**

  This optional parameter determines whether or not the application should send an email when it performs a successful deletion. If it is set to 0, no emails will be sent. If it is set to 1, and you have properly defined `-MAIL_CONFIG_PARAMS` and `-DELETE_EVENT_MAIL_SEND_PARAMS`, the application will send mail. By default, the parameter is set to 0.

- **-SEND_EMAIL_ON_MODIFY_FLAG**

This optional parameter determines whether or not the application should send an email when it performs a successful modification. If it is set to 0, no emails will be sent. If it is set to 1, and you have properly defined `-MAIL_CONFIG_PARAMS` and `-MODIFY_EVENT_MAIL_SEND_PARAMS`, the application will send mail. By default, the parameter is set to 0.

- **-SESSION_OBJECT**

  This optional parameter contains a reference to a *Session* object. Session is used to store information about the user in an otherwise stateless environment. Most notably, authentication information is stored in the session object so that the application can keep track of which users have logged in successfully and not bother them with logon screens every time new form is submitted.

- **-VALID_VIEWS**

  This is a required parameter that contains a reference to a list of valid views. For security reasons, this application will not display any view that has not specifically been named in this parameter. Thus, if you add a new view, you will have to add it to this parameter.

- **-VIEW_DISPLAY_PARAMS**

  This required parameter contains a reference to a set of view parameters. This parameter specifies the content that is published to all views. Views selectively subscribe to whatever elements of this parameter that they are interested in. If you wish to change something about how the data is displayed, it is a good chance that you will use this parameter rather than modifying the application code.

- **-VIEW_FILTERS_CONFIG_PARAMS**

  This optional parameter contains a reference to a list of *Extropia::Filter* configuration parameters.

- **-VIEW_LOADER**

  This required parameter contains a reference to a view loader.

2.8 Step 7: Modify Application Look-and-Feel

As we mentioned earlier, all views are located in the *webdb/Views/eXtropia* directory. If you want to change the way the user interface looks, you will have to edit one or more of these files. The eXtropia Applications Guide provides a detailed discussion of how to edit views. This guide also contains a section on typical configuration by example configuration changes which includes a typical change to the views.

2.9 Step 8: Run the Application

Running the application is a simple matter of pointing your web browser at the newly installed application executable using a URL such as:

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

Hopefully, the application will come up and you can begin to test it. If however, you receive an error message, you should consult Steps 8 and 9 in the *Installation* chapter of the eXtropia Applications Guide for help with debugging the problem.

2.10 Step 9: Debug the Application

Debugging can be quite an intensive process. Fortunately, there is an entire section to teach you how to debug CGI applications (and eXtropia applications in particular) in the eXtropia Applications Guide discussed in *Appendix A: Further References*.

2.11 Step 10: Review the Security Checklist

Have you read the section on security in the eXtropia Applications Guide? If not, you should do so now! Otherwise, here is the checklist (taken from this guide) of issues you should review before making this application available to the public.

1. **Add index.html files to all directories.**
2. **Change default filenames.**
3. **Use the .cgi extension for administrative or data files.**
4. **Move all files (except for executables) out of the web document tree altogether.**
5. **Set permissions correctly.**
6. **Quarantine writable files.**
7. **Revoke CGI rights except where required.**
8. **Disable SSI if possible.**
9. **Do not add insecure code.**
10. **Don’t disable built-in Perl security.**
11. **Stay informed by returning regularly to http://www.extropia.com/ or by adding yourself to the mailing list at register@extropia.com.**

2.12 Step 11: Test the Application

Before you actually go live with your application, you should also submit the application to thorough testing by a selected group of beta users. Some testing recommendations are presented in the eXtropia Applications Guide as a starting point.

2.13 Step 12: Register the Application

Finally, to make sure that you stay up to date with bug fixes, enhancements, and security bulletins, register your installation with eXtropia by sending an email to register@extropia.com. Include the URL of your installation in the email if you wish it to be added to the list of example sites.

Also, if you have made significant additions, please submit your ideas and solutions to the Radical Hacks Page (http://www.extropia.com/hacks/) at eXtropia by sending an email to hacks@extropia.com.

2.14 Installation Summary

Congratulations! You have installed an eXtropia application. If this is your first one, feel heartened by the fact that since all the applications are similar, installing your second one will be far easier.

But before you move on to your next one, perhaps you would like to learn more about how the application works. Alternatively, perhaps you want to modify the application in ways more drastic than simply changing the configuration parameters.

If that is the case, you will want to read the next half of this chapter. However, if you are not into the techie bits, you can just stop here. If you are happy with the application, there is no great need to spend time with the code. We realize that you may have better things to do, like run an online business!

;o)

3 WebDB Code Walkthrough

3.1 Overview

Finally, let’s take a look at how WebDB actually works.

As we mentioned in the introduction, the core WebDB is not contained in the CGI script. The script itself is simply a collection of configuration variables which are passed to the application object. This application object, *WebDB.pm*, is the actual workhorse.

With that in mind, we’ll focus primarily in walking through *WebDB.pm*. *WebDB.pm* has several methods through which it does its work.

> **Note that when we refer to *webdb.cgi* we are actually referring to any application executable that calls the *WebDB.pm* object (eg *address_book.cgi*).**

- **The new() Method**

  Public method that constructs the object and sets all incoming configuration parameters. The method is called from *webdb.cgi* and returns the application object. The method takes a single array of configuration parameters that are defined in `@APPLICATION_SETUP` in *webdb.cgi*.

- **The do() Method**

  Public method that specifies the main application workflow. The method is called from *webdb.cgi*, takes no parameters, and has no return value.

- **The _processAddRequest() Method**

  This protected method implements the workflow surrounding an add event.

- **The _processDeleteRequest() Method**

  This protected method implements the workflow surrounding a delete event.

- **The _processModifyRequest() Method**

  This protected method implements the workflow surrounding a modify event.

- **The _sendReceipt() Method**

  Protected method that performs various mailings using Extropia::Mail.

- **The _loadData() Method**

  Protected method that loads the results of a search on the data source. It returns an *Extropia::RecordSet*.

- **The _addRecord() Method**

This protected method performs the addition of a new record. The method is called from the `_processAddRequest()` method and returns `1` if successful, or `undef` if unsuccessful.

- **The _deleteRecord() Method**

  This protected method performs the deletion of a record. The method is called from the `_process-DeleteRequest()` method and returns `1` if successful, or `undef` if unsuccessful.

- **The _modifyRecord() Method**

  This protected method performs the modification of a record. The method is called from the `_processModifyRequest()` method and returns `1` if successful, or `undef` if unsuccessful.

3.2 The Application Preamble

Like all eXtropia application objects, *WebDB.pm* opens with the standard eXtropia application preamble that is documented in the *eXtropia Applications Guide*. But to refresh your memory, here is what the preamble contains:

- **Defines the package name.**
- **Loads supporting Perl modules.**
- **Declares inheritance from Extropia::App.**
- **Loads supporting eXtropia modules.**

The application has several standard Perl module dependencies including the following:

- **strict.pm**
- **Carp.pm**
- **vars.pm**

The application also has the following standard eXtropia module dependencies:

- **Extropia::DataSource**
- **Extropia::AuthManager**
- **Extropia::Mail**
- **Extropia::Log**
- **Extropia::App**

3.3 The new() Method

Like every eXtropia application object, this application also defines a `new()` method that is called by *webdb.cgi* to construct the application object. The method is public and returns the application object.

Essentially the method performs four steps:

1. **Check incoming variables**
2. **Create a Log object and AuthManager object**
3. **Check that optional parameters make sense**
4. **Return object**

3.3.1 Step 1: Check Incoming Variables

The `new()` method uses the `_rearrangeAsHash()` method from *Extropia::Base* to order the incoming parameters and to check that all required parameters have been submitted. It then used the `_dieIfRemainingParamsExist()` method, also from *Extropia::Base*, to make sure that no other parameters were submitted. This protects users who make typos in configuring the application.

The newly ordered and checked parameters are then added to the `$params` hash.

Next, where appropriate, optional parameters are assigned default values using the `_assignDefaults()` method in *Extropia::Base*.

Finally, defaults are overridden by incoming variables where appropriate. In particular, we allow the `view`, `max_records_per_page`, `sort_field1`, and `sort_field2` parameters to be overridden by user-defined values. We also assign values for administrative pagination variables.

```
sub new {
    my $package = shift;
    my $self;
    ($self, @_) = _rearrangeAsHash([
        -CGI_OBJECT,
        -SESSION_OBJECT,
        -DATASOURCE_CONFIG_PARAMS,
        -AUTH_MANAGER_CONFIG_PARAMS,
        -ADD_FORM_DH_MANAGER_CONFIG_PARAMS,
        -MODIFY_FORM_DH_MANAGER_CONFIG_PARAMS,
        -DELETE_FORM_DH_MANAGER_CONFIG_PARAMS,
        -LOG_CONFIG_PARAMS,
        -DEFAULT_SORT_FIELD1,
        -DEFAULT_SORT_FIELD2,
        -DEFAULT_VIEW_NAME,
        -DEFAULT_MAX_RECORDS_PER_PAGE,
        -REQUIRE_AUTH_FOR_SEARCHING_FLAG,
        -ALLOW_ADDITIONS_FLAG,
        -ALLOW_MODIFICATIONS_FLAG,
        -ALLOW_DELETIONS_FLAG,
        -VIEW_FILTERS_CONFIG_PARAMS,
        -VIEW_LOADER,
        -VALID_VIEWS,
        -VIEW_DISPLAY_PARAMS,
        -BASIC_DATA_VIEW_NAME,
        -MODIFY_FORM_VIEW_NAME,
        -DELETE_FORM_VIEW_NAME,
        -ADD_FORM_VIEW_NAME,
        -MAIL_CONFIG_PARAMS,
        -SEND_EMAIL_ON_DELETE_FLAG,
        -SEND_EMAIL_ON_MODIFY_FLAG,
        -SEND_EMAIL_ON_ADD_FLAG,
        -DELETE_EVENT_MAIL_SEND_PARAMS,
        -MODIFY_EVENT_MAIL_SEND_PARAMS,
        -ADD_EVENT_MAIL_SEND_PARAMS,
        -DELETE_EMAIL_BODY_VIEW,
        -ADD_EMAIL_BODY_VIEW,
        -MODIFY_EMAIL_BODY_VIEW
            ],
            [
        -CGI_OBJECT,
        -DATASOURCE_CONFIG_PARAMS,
        -DEFAULT_VIEW_NAME,
        -BASIC_DATA_VIEW_NAME,
        -MODIFY_FORM_VIEW_NAME,
        -DELETE_FORM_VIEW_NAME,
        -ADD_FORM_VIEW_NAME,
        -VALID_VIEWS,
        -VIEW_DISPLAY_PARAMS,
        -VIEW_LOADER
            ],
        @_);

    _dieIfRemainingParamsExist(@_);

    $self = _assignDefaults($self, {
        -SEND_EMAIL_ON_DELETE_FLAG         => 0,
        -SEND_EMAIL_ON_ADD_FLAG            => 0,
        -SEND_EMAIL_ON_MODIFY_FLAG         => 0,
        -ALLOW_ADDITIONS_FLAG              => 0,
        -ALLOW_MODIFICATIONS_FLAG          => 0,
        -ALLOW_DELETIONS_FLAG              => 0,
        -REQUIRE_AUTH_FOR_SEARCHING_FLAG   => 1,
        -DEFAULT_MAX_RECORDS_PER_PAGE      => 10
    });

    $self->{’_view_name’} = $self->{’-DEFAULT_VIEW_NAME’};
    my $cgi = $self->{’-CGI_OBJECT’};

    $self->{’_sort_field1’} = $cgi->param(’sort_field1’) ||
                           $self->{’-DEFAULT_SORT_FIELD1’};
        $self->{'_sort_field2'} = $cgi->param('sort_field2') ||
                                  $self->{'-DEFAULT_SORT_FIELD2'};

        $self->{'_max_records_per_page'} = $cgi->param('records_per_page') ||
                                        $self->{'-DEFAULT_MAX_RECORDS_PER_PAGE'};
```

```
        $self->{'_last_record_retrieved'} =
            $cgi->param('first_record_to_display') ||
                                           0;
        $self->{'_first_record_on_page'}  =
            $cgi->param('first_record_to_display') ||
                                           0;
        $self->{'_simple_search_string'}  =
            $cgi->param('simple_search_string') || "";
```

3.3.2 Step 2: Create a Log Object and AuthManager Object

Next, *Extropia::Log* and *Extropia::AuthManger* objects are created and added to the `$self` hash.

```
    $self->{'_auth_manager_object'} =  Extropia::AuthManager->create(
        @{$self->{'-AUTH_MANAGER_CONFIG_PARAMS'}}
    ) or die("Whoopsy!  I was unable to construct the " .
             "Authentication object in the new() method of MLM.pm. Please ".
             "contact the webmaster.");

    $self->{'_log'} =
        Extropia::Log->create(@{$self->{'-LOG_CONFIG_PARAMS'}})
        or die("Whoopsy!  I was unable to construct the " .
               "Log object in the new() method of MLM.pm. Please " .
               "contact the webmaster.");
```

3.3.3 Step 3: Check that optional parameters make sense

Then, the method checks to make sure that all optional parameters that have dependencies on other optional parameters are checked.

For example, if the optional `-SEND_EMAIL_ON_ADD_FLAG` is set to 1, the application will require that the optional `-MAIL_CONFIG_PARAMS` and `-ADD_EVENT_MAIL_SEND_PARAMS` parameters are defined as well.

```
    if ($self->{'-SEND_EMAIL_ON_MODIFY_FLAG'}) {
        if (!$self->{'-MAIL_CONFIG_PARAMS'} ||
            !$self->{'-MODIFY_EVENT_MAIL_SEND_PARAMS'}) {
            die("Whoopsy! In order to send mail, you must specify " .
                "-MAIL_CONFIG_PARAMS and -MODIFY_EVENT_MAIL_SEND_PARAMS");
        }
    }

    if ($self->{'-SEND_EMAIL_ON_ADD_FLAG'}) {
        if (!$self->{'-MAIL_CONFIG_PARAMS'} ||
            !$self->{'-ADD_EVENT_MAIL_SEND_PARAMS'}) {
            die("Whoopsy! In order to send mail, you must specify " .
                "-MAIL_CONFIG_PARAMS and -ADD_EVENT_MAIL_SEND_PARAMS");
        }
    }

    if ($self->{'-SEND_EMAIL_ON_DELETE_FLAG'}) {
        if (!$self->{'-MAIL_CONFIG_PARAMS'} ||
            !$self->{'-DELETE_EVENT_MAIL_SEND_PARAMS'}) {
            die("Whoopsy! In order to send mail, you must specify " .
                "-MAIL_CONFIG_PARAMS and -DELETE_EVENT_MAIL_SEND_PARAMS");
        }
    }
```

3.3.4 Step 4: Return the object

Finally the object itself is returned to the caller.

```
    return bless $self, $package;
```

3.4 The do() Method

Like every eXtropia application object, this application defines a `do()` method. The method is public and returns whatever view is required based upon the results of processing. It is called from the last line of *webdb.cgi*.

The `do()` method handles eight cases.

1. **User submits an Add request.**
2. **User submits a Modify request.**
3. **User submits a Delete request.**
4. **User requests to see the Add form**
5. **User requests to see the Modify form**

6. **user requests to see the Basic Data view.**

7. **User requests to see another view (such as BottomFrameView).**

3.4.1 Case 1: User Is Submitting An Add Request

If the user had just been on the *Add* form and submitted an add request, the value of the submit button named submit_add_record will have some value. As was discussed in the *eXtropia Applications Guide*, the following if control statement tests to see if the submit button has been clicked:

```
if ($cgi->param('submit_add_record') &&
       $self->{'-ALLOW_ADDITIONS_FLAG'}) {
```

If the tests succeed, the `do()` method processes the addition using the `_processAddRequest()` method discussed later.

If the processing is successful, it calls the `_displayView()` method to display the *Basic Data* view. If, on the other hand, the processing is unsuccessful (most likely due to data handling errors), the `do()` method redisplays the *Add Form* view using the `_displayView()` method.

```
if ($cgi->param('submit_add_record') &&
       $self->{'-ALLOW_ADDITIONS_FLAG'}) {
    if ($auth_manager) {
        $auth_manager->authenticate();
    }

    my $add_request_success = $self->_processAddRequest(
        -CGI_OBJECT                 => $self->{'-CGI_OBJECT'},
        -LOG_OBJECT                 => $self->{'_log_object'},
        -DATASOURCE_CONFIG_PARAMS   =>
            $self->{'-DATASOURCE_CONFIG_PARAMS'},
        -MAIL_CONFIG_PARAMS         => $self->{'-MAIL_CONFIG_PARAMS'},
        -DATA_HANDLER_CONFIG_PARAMS =>
            $self->{'-ADD_FORM_DH_MANAGER_CONFIG_PARAMS'},
        -EMAIL_BODY_VIEW            => $self->{'-ADD_EMAIL_BODY_VIEW'},
        -MAIL_SEND_PARAMS           =>
            $self->{'-ADD_EVENT_MAIL_SEND_PARAMS'},
        -SEND_EMAIL_FLAG            =>
            $self->{'-SEND_EMAIL_ON_ADD_FLAG'},
        -VIEW_DISPLAY_PARAMS        => $self->{'-VIEW_DISPLAY_PARAMS'},
        -VIEW_LOADER                => $self->{'-VIEW_LOADER'}
    );

    if (!$add_request_success) {
        $self->{'_view_name'} = $self->{'-ADD_FORM_VIEW_NAME'};
    }
    else {
        $self->{'_view_name'} = $self->{'-BASIC_DATA_VIEW_NAME'};
    }

   return $self->_displayView(
        -VIEW_NAME                  => $self->{'_view_name'},
        -LOAD_DATA_FLAG             => 1,
        @default_view_display_params
    );
}
```

3.4.2 Case 2: User Is Submitting A Modify Request

If you look at the code used to modify a record, you will see that it is almost identical to the code used for additions.

```
    elsif ($cgi->param('submit_modify_record') &&
        $self->{'-ALLOW_MODIFICATIONS_FLAG'}) {

        if ($auth_manager) {
            $auth_manager->authenticate();
        }

        my $modify_request_success = $self->_processModifyRequest(
            -CGI_OBJECT                 => $self->{'-CGI_OBJECT'},
            -LOG_OBJECT                 => $self->{'_log_object'},
            -DATASOURCE_CONFIG_PARAMS   =>
                $self->{'-DATASOURCE_CONFIG_PARAMS'},
            -MAIL_CONFIG_PARAMS         => $self->{'-MAIL_CONFIG_PARAMS'},
            -DATA_HANDLER_CONFIG_PARAMS =>
                $self->{'-MODIFY_FORM_DH_MANAGER_CONFIG_PARAMS'},
            -EMAIL_BODY_VIEW            =>
                $self->{'-MODIFY_EMAIL_BODY_VIEW'},
            -MAIL_SEND_PARAMS           =>
                $self->{'-MODIFY_EVENT_MAIL_SEND_PARAMS'},
            -SEND_EMAIL_FLAG            =>
                $self->{'-SEND_EMAIL_ON_MODIFY_FLAG'},
            -VIEW_DISPLAY_PARAMS        => $self->{'-VIEW_DISPLAY_PARAMS'},
            -VIEW_LOADER                => $self->{'-VIEW_LOADER'}
        );

        if (!$modify_request_success) {
            $self->{'_view_name'} = $self->{'-MODIFY_FORM_VIEW_NAME'};
        }
        else {
            $self->{'_view_name'} = $self->{'-BASIC_DATA_VIEW_NAME'};
        }

        return $self->_displayView(
            -VIEW_NAME                  => $self->{'_view_name'},
            -LOAD_DATA_FLAG             => 1,
            @default_view_display_params
        );
    }
```

3.4.3 Case 3: User Is Submitting A Delete Request

If you look at the code used to modify a record, you will see that it is almost identical to the code used for additions.

```
    elsif ($cgi->param('submit_delete_record') &&
        $self->{'-ALLOW_DELETIONS_FLAG'}) {
```

```
        if ($auth_manager) {
            $auth_manager->authenticate();
        }

        my $delete_request_success = $self->_processDeleteRequest(
            -CGI_OBJECT                 => $self->{'-CGI_OBJECT'},
            -LOG_OBJECT                 => $self->{'_log_object'},
            -DATASOURCE_CONFIG_PARAMS   =>
                $self->{'-DATASOURCE_CONFIG_PARAMS'},
            -MAIL_CONFIG_PARAMS         => $self->{'-MAIL_CONFIG_PARAMS'},
            -DATA_HANDLER_CONFIG_PARAMS =>
                $self->{'-DELETE_FORM_DH_MANAGER_CONFIG_PARAMS'},
            -EMAIL_BODY_VIEW            =>
                $self->{'-DELETE_EMAIL_BODY_VIEW'},
            -MAIL_SEND_PARAMS           =>
                $self->{'-DELETE_EVENT_MAIL_SEND_PARAMS'},
            -SEND_EMAIL_FLAG            =>
                $self->{'-SEND_EMAIL_ON_DELETE_FLAG'},
            -VIEW_DISPLAY_PARAMS        => $self->{'-VIEW_DISPLAY_PARAMS'},
            -VIEW_LOADER                => $self->{'-VIEW_LOADER'}
        );

        if (!$delete_request_success) {
            $self->{'_view_name'} = $self->{'-DELETE_FORM_VIEW_NAME'};
        }
        else {
            $self->{'_view_name'} = $self->{'-BASIC_DATA_VIEW_NAME'};
        }

       return $self->_displayView(
            -VIEW_NAME                  => $self->{'_view_name'},
            -LOAD_DATA_FLAG             => 1,
            @default_view_display_params
        );
    }
```

3.4.4 Case 4: User Requests To see The Add Form

If the user has requested to see the *Add* form, the method uses the `_displayView()` method to display the view.

```
    elsif ($cgi->param('display_add_form') &&
           $self->{'-ALLOW_ADDITIONS_FLAG'}) {

        if ($auth_manager) {
            $auth_manager->authenticate();
        }

        return $self->_displayView(
            -VIEW_NAME                  => $self->{'-ADD_FORM_VIEW_NAME'},
            @default_view_display_params
        );
    }
```

3.4.5 Case 5: User Requests To See The Modify Form

If the user has requested to see the *Modify* form, the method uses the `_displayView()` method to display the view.

```
elsif ($cgi->param('display_modification_form') &&
       $self->{'-ALLOW_MODIFICATIONS_FLAG'}) {

    if ($auth_manager) {
        $auth_manager->authenticate();
    }

    return $self->_displayView(
        -VIEW_NAME                  => $self->{'-MODIFY_FORM_VIEW_NAME'},
        -LOAD_DATA_FLAG             => 1,
        @default_view_display_params
    );
}
```

3.4.6 Case 6: User Requests To See The Basic Data View

If the user has requested to see the *Basic data* view, the method will use the `_displayView()` method to display the view

```
elsif ($cgi->param('display_data_view') ||
       ($self->{'_view_name'} eq $self->{'-BASIC_DATA_VIEW_NAME'})) {

    if ($self->{'-REQUIRE_AUTH_FOR_SEARCHING_FLAG'}) {
        if ($auth_manager) {
            $auth_manager->authenticate();
        }
    }

   return $self->_displayView(
        -VIEW_NAME                  => $self->{'-BASIC_DATA_VIEW_NAME'},
        -LOAD_DATA_FLAG             => 1,
        @default_view_display_params
    );
}
```

3.4.7 Case 7: User Requests To See Another View

If the user has requested to see another view, the method will use the `_displayView()` method to display the view.

```
    else {
        return $self->_displayView(
            -VIEW_NAME  => $cgi->param('view') ||
                           $self->{'_view_name'},
            @default_view_display_params
        );
    }
```

There are several possible views that can be displayed. Which view is actually displayed depends on the rules explained in the following list:

- **No view specified.**

  The default view is displayed. This view is specified in *webdb.cgi* using @APPLICATION_SETUP.

- **The Add button was clicked from the BasicDataView view.**

  The *Add* form is displayed.

- **The Delete button was clicked from the BasicDataView view.**

  The *Delete* form is displayed.

- **The Modify button was clicked from the BasicDataView view.**

  The *Modify* form is displayed.

- **The user submits the Add form successfully.**

  The item is added and the *BasicDataView* view is displayed.

- **The user submits the Add form but there is a data handler error.**

  The *Add* form is again displayed except this time the error message is included.

- **The user submits the Add form but there is a problem performing the addition.**

  The *Add* form is again displayed except this time the error message is included.

- **The user submits the Delete form successfully.**

  The item is deleted and the *BasicDataView* view is displayed.

- **The user submits the Delete form but there is a data handler error.**

  The *Delete* form is again displayed except this time the error message is included.

- **The user submits the Delete form but there is a problem performing the deletion.**

The *Delete* form is again displayed except this time the error message is included.

- **The user clicks the Power Search Button.**

  The *Power Search* button on the Power Search form is clicked.

  The *BasicDataView* view is displayed showing the results of the search.

  The *BasicDataView* view is displayed showing the results of the search.

- **The user clicks the View as Defined Button.**

  The *BasicDataView* view is displayed showing the results of the search.

3.5 The _displayView() Method

This method is used to display the requested view. Essentially, the method performs four functions:

1. **Check incoming variables.**
2. **Load data if required to do so.**
3. **Check for errors.**
4. **Display view.**

3.5.1 Step 1: Check Incoming Variables

The method uses the `_rearrangeAsHash()` method from *Extropia::Base* to order the incoming parameters and to check that all required parameters have been submitted. It then uses the `_dieIfRemainingParamsExist()` method, also from *Extropia::Base*, to check that no other parameters were submitted. This is protects users who make typos in configuring the application.

The newly ordered and checked parameters are then added to the `$params` hash.

Finally, where appropriate, optional parameters are assigned default values using the `_assignDefaults()` method in *Extropia::Base*.

```
    sub _displayView {
        my $self = shift;
        my ($params) = _rearrangeAsHash([
            -AUTH_MANAGER,
            -SESSION_OBJECT,
            -LOG_OBJECT,
            -MAX_RECORDS_PER_PAGE,
            -FIRST_RECORD_ON_PAGE,
            -SORT_FIELD1,
            -SORT_FIELD2,
            -SIMPLE_SEARCH_STRING,
            -VIEW_NAME,
            -VALID_VIEWS,
            -VIEW_LOADER,
            -VIEW_FILTERS_CONFIG_PARAMS,
            -VIEW_DISPLAY_PARAMS,
            -DATASOURCE_CONFIG_PARAMS,
            -LAST_RECORD_RETRIEVED,
            -CGI_OBJECT,
            -LOAD_DATA_FLAG
                ],
                [
            -MAX_RECORDS_PER_PAGE,
            -FIRST_RECORD_ON_PAGE,
            -SORT_FIELD1,
            -SORT_FIELD2,
            -SIMPLE_SEARCH_STRING,
            -VIEW_NAME,
            -VALID_VIEWS,
            -VIEW_LOADER,
            -VIEW_DISPLAY_PARAMS,
                ],
            @_
        );

        my $record_set;

        $params = _assignDefaults($params, {
            -LOAD_DATA_FLAG          => 0,
            -SIMPLE_SEARCH_STRING    => "",
            -SORT_FIELD1             => "",
            -SORT_FIELD2             => "",
            -MAX_RECORDS_PER_PAGE    => 10,
            -LAST_RECORD_RETRIEVED   => 0
        });

        if ($params->{'-LOAD_DATA_FLAG'}) {
            if (!$params->{'-DATASOURCE_CONFIG_PARAMS'} ||
                !$params->{'-CGI_OBJECT'}) {
                die("Whoopsy! -DATASOURCE_CONFIG_PARAMS and " .
                    "-CGI_OBJECT are required by _displayView(), " .
                    "but have not been supplied.");
        }
```

3.5.2 Step 2: Load Data If Required

Next, the method loads the data source entries if it is required to do so.

```
if ($params->{'-LOAD_DATA_FLAG'}) {
    if (!$params->{'-DATASOURCE_CONFIG_PARAMS'} ||
        !$params->{'-CGI_OBJECT'}) {
        die("Whoopsy! -DATASOURCE_CONFIG_PARAMS and " .
            "-CGI_OBJECT are required by _displayView(), " .
            "but have not been supplied.");
    }

    $record_set = $self->_loadData((
        -DATASOURCE_CONFIG_PARAMS  =>
             $params->{'-DATASOURCE_CONFIG_PARAMS'},
        -SORT_FIELD1               => $params->{'-SORT_FIELD1'},
        -SORT_FIELD2               => $params->{'-SORT_FIELD2'},
        -MAX_RECORDS_PER_PAGE      => $params->{'-MAX_RECORDS_PER_PAGE'},
        -LAST_RECORD_RETRIEVED     =>
             $params->{'-LAST_RECORD_RETRIEVED'},
        -SIMPLE_SEARCH_STRING      => $params->{'-SIMPLE_SEARCH_STRING'},
        -CGI_OBJECT                => $params->{'-CGI_OBJECT'}
    ));
}
```

3.5.3 Step 3: Check For Errors

Next the method checks to see if any errors have been built up in the processing of user input. To do so, it accesses the error object provided by *Extropia::Base* and goes through each error extracting the message that it will then pass to all views.

```
my $errors_ref = $self->getErrors();

my @errors;
my $error;
foreach $error (@$errors_ref) {
    push (@errors, $error->getMessage());
}
```

3.5.4 Step 4: Display View

Finally, the method uses the `_loadViewAndDisplay()` method from *Extropia::App* to return the requested view.

```
    my $view_display_params = $params->{'-VIEW_DISPLAY_PARAMS'};

    return $self->_loadViewAndDisplay((
        -RECORD_SET                  => $record_set,
        -ERROR_MESSAGES              => \@errors,
        -AUTH_MANAGER                => $params->{'-AUTH_MANAGER'},
        -SESSION_OBJECT              => $params->{'-SESSION_OBJECT'},
        -LOG_OBJECT                  => $params->{'-LOG_OBJECT'},
        -MAX_RECORDS_PER_PAGE        => $params->{'-MAX_RECORDS_PER_PAGE'},
        -FIRST_RECORD_ON_PAGE        => $params->{'-FIRST_RECORD_ON_PAGE'},
        -SORT_FIELD1                 => $params->{'-SORT_FIELD1'},
        -SORT_FIELD2                 => $params->{'-SORT_FIELD2'},
        -SIMPLE_SEARCH_STRING        => $params->{'-SIMPLE_SEARCH_STRING'},
        -VIEW_NAME                   => $params->{'-VIEW_NAME'},
        -VALID_VIEWS                 => $params->{'-VALID_VIEWS'},
        -VIEW_LOADER                 => $params->{'-VIEW_LOADER'},
        -VIEW_FILTERS_CONFIG_PARAMS =>
            $params->{'-VIEW_FILTERS_CONFIG_PARAMS'},
        @$view_display_params
        ));
    }
```

3.6 The _processAddRequest() Method

There are four steps involved with processing an addition:

1. **Check incoming variables**
2. **Handle Data**
3. **Add the record**
4. **Send Mail**

3.6.1 Step 1: Check Incoming Variables

The method uses the `_rearrangeAsHash()` method from *Extropia::Base* to order the incoming parameters and to check that all required parameters have been submitted. It then uses the `_dieIfRemainingParamsExist()` method, also from *Extropia::Base*, to check that no other parameters were submitted. This is protects users who make typos in configuring the application.

The newly ordered and checked parameters are then added to the `$params` hash.

```
    sub _processAddRequest {
        my $self = shift;

        my ($params) = _rearrangeAsHash([
            -CGI_OBJECT,
            -LOG_OBJECT,
            -DATASOURCE_CONFIG_PARAMS,
            -MAIL_CONFIG_PARAMS,
            -DATA_HANDLER_CONFIG_PARAMS,
            -EMAIL_BODY_VIEW,
            -MAIL_SEND_PARAMS,
            -SEND_EMAIL_FLAG,
            -VIEW_DISPLAY_PARAMS,
            -VIEW_LOADER
                ],
                [
            -CGI_OBJECT,
            -DATASOURCE_CONFIG_PARAMS,
            -VIEW_DISPLAY_PARAMS,
            -VIEW_LOADER
                ],
            @_
        );
```

3.6.2 Step 2: Handle the Data

The next step in processing a form is to submit the incoming data to the data handling rules contained in the data handler manager. To handle the incoming data, we use the `_handleIncomingData()` method in *Extropia::App*. This method checks that incoming data conforms to the rules specified in the data handler manager defined in mlm.cgi. The method is discussed in detail in the *eXtropia Applications Guide* referenced in Appendix A: Further References.

If the incoming form data passes inspection, the method returns a `1` (true).

If, on the other hand, the data does not pass, the method returns an `undef` and logs the error using an *Extropia::Log* object if one has been supplied.

In the case of a data handling error, we also populate the local error variable given to us by *Extropia::Base* using the `addError()` method also inherited from *Extropia::Base*. The errors added are returned from the `_getDataHandlerErrors()` method defined in *Extropia::App*. Finally, we return `undef` to the `do()` method to let it know that the form data was bad.

```
        my $data_handler_success = $self->_handleIncomingData(
            -CGI_OBJECT                  => $cgi,
            -LOG_OBJECT                  => $log,
            -DATA_HANDLER_CONFIG_PARAMS =>
                $delete_form_dhm_config_params
        );

        if (!$data_handler_success) {
            my $error;
            foreach $error ($self->_getDataHandlerErrors()) {
                $self->addError($error);
            }
            $self->{-VIEW}           = $add_form_view_name;
        }
```

3.6.3 Step 3: Adding the Record

If the data handler manager reports success, the method continues to process the request. The act of adding a new record is performed in the `_addRecord()` method that is discussed later in this chapter.

The `_addRecord()` method returns a `1` (true) if it is successful. If it is not successful, it returns an `undef` and logs the problem. If the addition is unsuccessful, the method sets the `-VIEW` configuration parameter to the *Add* form so that the form will be redisplayed with errors so that the user can try again.

```
    my $addition_success = $self->_addRecord((
        -CGI_OBJECT                => $params->{'-CGI_OBJECT'},
        -LOG_OBJECT                => $params->{'-LOG_OBJECT'},
        -DATASOURCE_CONFIG_PARAMS => $params->{'-DATASOURCE_CONFIG_PARAMS'}
    ));

    if (!$addition_success) {
        return undef;
    }
```

3.6.4 Step 4: Sending Mail

If, on the other hand, the record addition was successful, the method continues to process the request. Recall that in the application executable we specified a set of `-MAIL_SEND_PARAMS` that corresponded to the parameters that should be used in email sent out for different circumstances.

Here is where those configuration parameters are used.

The method begins by dealing with the admin receipt. Initially, it checks to see if the administrator has configured the application to send email receipts at all. Recall that we defined `-SEND_MAIL_ON_DELETE_FLAG` in the application executable. If set to 1, the method knows to send the user a receipt.

Mailing involves three steps:

- **Get the body of the mail using the specified view object.**
- **Instantiate a mailer.**
- **Send the mail using the _sendReceipt() method discussed later in this chapter.**

Once the email receipt has been sent (or not depending on how you have configured the application), the method sends the user a notification. Of course, as was the case with the administrator receipt, whether or not the method sends the user notification will depend on the `-SEND_USER_RECEIPT_FLAG_ON_DELETE` configuration parameter.

The process of emailing the administrator notification is exactly the same as for emailing the user receipt so rather than bore you, well just present the code in the following example:

```
    my $view_loader = $params->{'-VIEW_LOADER'};

    if ($params->{'-SEND_EMAIL_FLAG'}) {
        my $view = $view_loader->create(
            $params->{'-EMAIL_BODY_VIEW'}
        );
```

```
        my $body = $view->display(
            @{$params->{'-VIEW_DISPLAY_PARAMS'}}
        );

        $self->_sendReceipt((
            -MAIL_CONFIG_PARAMS => $params->{'-MAIL_CONFIG_PARAMS'},
            -BODY               => $body,
            @{$params->{'-MAIL_SEND_PARAMS'}}
        ));
    }
```

3.7 The _processModifyRequest() Method

This method does exactly the same thing as the `_processAddRequest()` method except that it is specific to modifications rather than additions. Because it works exactly the same, we wont bother listing the code here.

3.8 The _processDeleteRequest() Method

This method does exactly the same thing as the `_processAddRequest()` method except that it is specific to deletions rather than additions. Because it works exactly the same, we wont bother listing the code here.

3.9 The _sendReceipt() Method

This protected method is called from within each of the three `_processXXXRequest()` methods and is used to perform mailings. The method performs the following functions:

1. **Define variables.**
2. **Instantiate an Extropia::Mail mailer.**
3. **Attempt to send the requested mail.**

3.9.1 Step 1: Define Variables

The method begins by using the `_rearrange()` method from Extropia::Base to order incoming parameters and check to make sure that all the required parameters have values. It then shifts the ordered variables off the parameter list.

```
    sub _sendReceipt {
        my $self = shift;
        @_ = _rearrange([
            -MAIL_CONFIG_PARAMS,
            -FROM,
            -TO,
            -SUBJECT,
            -BODY,
                ],
                [
            -MAIL_CONFIG_PARAMS,
            -FROM,
            -TO,
            -SUBJECT,
            -BODY
                ],
            @_
        );

        my $mail_config_params_ref = shift;
        my $from                   = shift;
        my $to_list_ref            = shift;
        my $subject                = shift;
        my $body                   = shift;
```

3.9.2 Step 2: Instantiate An Extropia::Mail Mailer.

The mail configuration parameters are then used to instantiate a mailer object.

```
    my $mailer = Extropia::Mail->create(@$mail_config_params_ref)
       or confess("Whoopsy!  I was unable to construct the Mail " .
                  "object in the _sendMail() method of MLM.pm. " .
                  "Please contact the webmaster.");
```

3.9.3 Step 3: Attempt to send the requested mail.

The mail is then sent using the `send()` method in the mailer object.

```
    return $mailer->send((
        -FROM    => $from,
        -TO      => $to_list_ref,
        -SUBJECT => $subject,
        -BODY    => $body
    ));
```

3.10 The _loadData() Method

This protected method is called from within the `do()` method and is used to load guestbook entries for display. The method performs the following functions:

1. **Define variables.**
2. **Refine search parameters based upon the simple_search_string and the display_without_admin_review variables.**
3. **Instantiate an Extropia::DataSource object.**
4. **Perform a search on the data source.**
5. **Populate the internal error parameter if the search returned nothing and return undef, or return the record set returned from the search.**

3.10.1 Step 1: Define The Variables

The method begins by using the `_rearrange()` method from *Extropia::Base* to order incoming parameters and check to make sure that all the required parameters have values.

```
    sub _loadData {
        my $self = shift;
        @_ = _rearrange([
            -DATASOURCE_CONFIG_PARAMS,
            -SORT_FIELD1,
            -SORT_FIELD2,
            -MAX_RECORDS_PER_PAGE,
            -LAST_RECORD_RETRIEVED,
            -SIMPLE_SEARCH_STRING
                ],
                [
            -DATASOURCE_CONFIG_PARAMS,
            -SORT_FIELD1,
            -SORT_FIELD2,
            -MAX_RECORDS_PER_PAGE,
            -LAST_RECORD_RETRIEVED,
            -SIMPLE_SEARCH_STRING
                ],
            @_);

        my $datasource_config_params_ref = shift;
        my $sort_field1                  = shift;
        my $sort_field2                  = shift;
        my $max_records_to_retrieve      = shift;
        my $last_record_retrieved        = shift;
        my $simple_search_string         = shift;
```

3.10.2 Step 2: Refine Search Parameters

Next, the method refines the search string.

```
    if ($simple_search_string) {
        $simple_search_string = "* =i ’*" . $simple_search_string . "*’";
    }
    else {
        $simple_search_string = "* =i ’*’";
    }
```

3.10.3 Step 3: Instantiate an Extropia::DataSource object.

The data source configuration parameters are then used to instantiate a data source object.

```
    my $search_ds = Extropia::DataSource->create(@$datasource_config_params_ref)
       or confess("Whoopsy!  I was unable to construct the " .
                  "DataSource object in the _loadData() method of " .
                  "WebDB.pm. Please contact the webmaster.");
```

3.10.4 Step 4: Perform a search on the data source.

Then, the data source is searched.

```
    return $search_ds->search(
        -ORDER                    => "$sort_field1, $sort_field2",
        -MAX_RECORDS_TO_RETRIEVE => $max_records_to_retrieve,
        -LAST_RECORD_RETRIEVED   => $last_record_retrieved,
        -SEARCH                  => $simple_search_string
   );
```

3.11 The _addRecord() Method

This protected method is called from the `_processAddRequest()` method and is used to add a new entry to the mailing list. The method performs the following functions:

1. **Define variables.**
2. **Instantiate an Extropia::DataSource object.**
3. **Add the record.**
4. **Log and report results.**

3.11.1 Step 1: Define variables

The method begins by using the `_rearrange()` method from *Extropia::Base* to order incoming parameters and check to make sure that all the required parameters have values.

```
    sub _addRecord {
        my $self = shift;
        @_ = _rearrange([
            -CGI_OBJECT,
            -LOG_OBJECT,
            -DATASOURCE_CONFIG_PARAMS
                ],
                [
            -CGI_OBJECT,
            -DATASOURCE_CONFIG_PARAMS
                ],
            @_);

        my $cgi                         = shift;
        my $log                         = shift;
        my $datasource_config_params_ref = shift;
```

3.11.2 Step 2: Instantiate an Extropia::DataSource object.

The data source configuration parameters are then used to instantiate a data source object.

```
my $add_ds = Extropia::DataSource->create(@$datasource_config_params_ref)
   or confess("Whoopsy!  I was unable to construct the " .
              "DataSource object in the _addRecord() method " .
              "of WebDB.pm. Please contact the webmaster.");
```

3.11.3 Step 3: Add the record

Then a hash of name/value pairs is created to use to add the record.

```
my @params = $cgi->param();
my (%add_hash, $param);

foreach $param (@params) {
    $add_hash{$param} = $cgi->param($param);
}

my $rows_added = $add_ds->add(-ADD => \%add_hash);
```

3.11.4 Step 4: Log and report results.

Finally, if there were any problems adding the record, the errors are logged and the method returns undef to report the problem to the caller. Otherwise, the success is logged and a true value (1) is returned to the caller to report success.

```
    if ($add_ds->getErrorCount()) {
        $self->addError($add_ds->getLastError());
        if ($log) {
            $log->log(
                 -SEVERITY =>  Extropia::Log::WARN,
                 -EVENT    => "FAILED_ADDITION|" .
                              $add_ds->getLastError()->getMessage()
            );
            return undef;
        }
    }

    else {
        if ($log) {
            my $key;
            my @add_array;
            foreach $key (keys %add_hash) {
               push (@add_array, "$key=" . $add_hash{$key});
            }
            my $add_string = join("\|", @add_array);
            $log->log(
                 -SEVERITY =>  Extropia::Log::INFO,
                 -EVENT    => "ADDITION PERFORMED\|" .
                              "ADD_DEFINITION: $add_string"
            );
        }
    }
    return 1;
```

3.12 The _deleteRecord() Method

This protected method is called from the `_processDeleteRequest()` method and is used to delete an entry from the mailing list. The method is almost identical to the `_addRecord()` method so we wont list the code here. Instead we will outline its functions:

- **Define variables.**
- **Generate a string to be used for deletions based upon the user input.**
- **Instantiate an Extropia::DataSource object.**
- **Attempt to delete the record.**
- **Log the results and return undef on errors.**
- **Return 1 if successful.**

3.13 The _modifyRecord() Method

This protected method is called from the `_processModifyRequest()` method and is used to delete an entry from the mailing list. The method is almost identical to the `_addRecord()` method so we won't list the code here. Instead we will outline its functions:

3.14 Code Walkthrough in Review

Hopefully that was not too much to take in. The code the makes the mailing list manager application work is actually not all that difficult. In summary, what you learned is that:

1. **The new() method instantiates the object.**
2. **The do() method handles the workflow.**

   In the case of this application, the workflow boils down to three cases including:

   1. **Allow a user to add a new entry**
   2. **Allow the user to delete an entry**
   3. **Allow an administrator to send a mailing to all list members**
3. **The _sendMail() method helps the do() method by sending the various mails the application needs to send.**
4. **The _addRecord() method helps the do() method by actually performing the new entry addition using Extropia::DataSource.**
5. **The _deleteRecord() method helps the do() method by actually performing deletions using Extropia::DataSource.**
6. **The _modifyRecord() method helps the do() method by actually performing modifications using Extropia::DataSource.**

;o)

4 Configuration by Example

4.1 Overview

Since most eXtropia applications are data source based, we have moved the configuration by example of this data source based application to the general eXtropia Application Guide described in *Appendix A: Further References*.

;o)

5 Appendix A: Further References

5.1 References

There are two primary references that supplement this guide.

- **eXtropia Applications Guide**

  This guide contains complete information about how eXtropia applications are installed and configured in general. Most eXtropia applications follow a similar design, so once you understand one application you can easily pick up all the others.

  For this reason, we heavily reference the eXtropia Applications Guide in the documentation for individual applications. Not only that, but the eXtropia Applications Guide contains a slew of information from our CGI tutorials that have evolved over the last six years.

  At the very least, we usually recommend that everyone at the very least always read the security section of this guide. We update this section of the guide as soon as any new Web security issues are discovered, so you should consider reading through this if you are not aware of all the security ramifications involved in web programming.

  The eXtropia Applications Guide can be found at the following URL:

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

- **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

5.2 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 (sbekman@stason.org)**

  Stas wrote the software that generated the HTML and PDF for this guide.

- **Chris Hughes (chris@fysh.org)**
- **Lyndon Drake (lyndon@kcbbs.gen.nz)**
- **Nikhil Kaul (nikhil.kaul@barclayscapital.com)**
- **Josh Hill (josh@extropia.com)**

  Josh helped edit and complete the documentation process.

5.3 Changes

No changes yet. Latest Documentation version is Sept. 3, 2001.

;o)

Table of Contents: