



Exclamation!
Desktop Web Publisher

Designer's Guide

24 October 2003

Ten League Boots™

Copyright © 2003 Ten League Boots. All rights reserved.

Comments and feedback welcome. Please e-mail support@tenleagueboots.com





Contents

Introduction to Exclamation	7
Exclamation Teamwork	7
How to avoid reading more than you need	9
Tools	10
Exclamation's view of the world	10
Scenarios for building a production Web site	15
A word about the examples used in this book	15
Creating a One-Page Web Site	19
Exclamation Web Sites	19
A word about Dreamweaver and GoLive Templates	20
Exclamation site structure	21
Content	22
Site Design	22
Preview	23
Creating a Web site	23
Creating the Content Declaration file	25
Creating a Template Declaration File	26
Filtering content using Keep and Omit	27
Creating the Template file	29
Using Exclamation to create the project	31
Rendering the Template	32
Multiple Output Files	33
Creating a Template with multiple output files	33

⋮

Creating the new Template file	35
Creating the second Template Declaration file	37
Rendering the new pages	38
Segmenting	39
Creating a simple segmented list	39
Creating a new Template Declaration File	41
Adding the new Template file	42
Rendering the new pages	43
A segment with multiple output files	44
Creating the index Template Declaration file	45
Amending the index Template file	46
Creating the star-rating page Template Declaration file	47
Creating the detail Template file	48
Conditional Commands	51
Vanish	51
Element	52
Content	52
Tag	53
Subtables	53
Between and beforelast	55
To if or to ifnot, that is the question	56
Advanced topics	56
Sorts and other live action on pages	57
List order in segmented lists	57
Publishing: Uploading and Viewing	59
Editing content	60
Editing the design	60
Generating the pages	60
Maintaining multiple designs	61
Previewing the results	62

Uploading to the hosting service 62
 Configuring Exclamation with hosting service information 63
 Uploading 64
Viewing the completed, hosted, Web site 64
Index 65

·
·
·



Introduction to Exclamation

Creating and maintaining an effective Web site is not easy. The best information comes from within an organization, but transforming that information into Web pages is error-prone, especially when it's done by someone outside the organization.

Meanwhile, keeping a Web site's information fresh and up-to-date can be tedious and time-consuming. Good site design (graphic style and navigation) requires a professional designer. Revising site structure and style can be prohibitively expensive. And ensuring information accuracy can be virtually impossible. Even an initially well-done site can quickly become a "cobweb site" that goes for months or even years without change, no longer serving either the site owner or the site visitors. When people revisit a site and find nothing new, they don't return again unless they have to—it's that simple.

The wide array of single-user Web-authoring applications all weld site information with site design, making it difficult and/or expensive to change either one. Although some of these applications provide features to ease the task of making global changes to the style of the pages, the content is still irreversibly connected to those pages, and can only be changed by editing the page directly. If the same text appears on multiple pages, it needs to be changed on each of them.

In summary, the professional single-user applications do not acknowledge the reality that an information-rich site requires domain knowledge, Web design expertise, and application skills, and it is seldom that one person has all of these credentials. At the other end of the scale, traditional content management systems, developed for large-scale information publishers like newspapers and large corporations, are far too expensive to design and implement to be of practical use to smaller organizations.

Exclamation Teamwork

Exclamation brings the power of a content management system (CMS) to user desktops at a small fraction of the cost and complexity of the enterprise-scale solutions. What distinguishes



a CMS from the typical desktop's hand-crafted approach is that a CMS automatically constructs a Web site from a database of site information (also known as "content") according to a site design (layout and structure). These are two distinctly different disciplines, and Exclamation separates its users accordingly.

The content is the responsibility of the site business manager and content creators. The site business manager is the person in charge of business purpose of the site. Typically this is the owner of the site. The site's business manager decides what information has to be in the site, and delegates the data preparation to the content contributor, who is someone they manage (or, in many cases the business manager will handle the day-to-day content changes themselves). Since the content is maintained separately from the source code, there is no need for the content contributor to have any HTML skills.

Layout and structure are handled by the second group, the Web designers and all those in their team: illustrators, developers, database engineers, and so on comprise the first group. Typically the business manager is a client of the Web designer.

The Exclamation tools enable the two groups to work cooperatively on designing the site, preparing the information content, and managing the deployment and evolution of the site over its lifetime, from initial conception through ongoing maintenance. Exclamation puts a strong emphasis on ease of use for all members of the Web site team:

- **Web Designer**—uses Web design applications (for example, Macromedia Dreamweaver or Adobe GoLive) to create the site visual design and navigation, prepare occasional hand-crafted pages, and develop the templates that will be used to generate content pages. The Web designer make changes to the actual Web site files.
- **Web Developer**—uses administration tools to run the Web server and maintain the system. Frequently the Web developer is also responsible for database applications (for example SQL). The Web developer is sometimes also the Web designer.

- Content Contributor—maintains the lists and tables of information that constitute the content, and uses standard Web browsers to preview the generated site.
- Site Manager—uses a simple Exclamation desktop application to review site status and publish (upload). The site manager is often also the content contributor.

Exclamation enables the Web designers and the content contributors to do their work on their individual desktops or laptops, occasionally connecting to an office network (LAN) or directly to the Internet to coordinate their individual tasks. This is especially important when site design is done by an independent Web Designer at a remote location, and also allows the content contributor to make changes to the content and publish them without further assistance.

How to avoid reading more than you need

If you really hate reading documentation stop right here. Open the movie sample project in Exclamation and point your favorite editor at the `Site Design/onepage` folder and start playing with the files. Use the *SongBird Designer's Reference Manual* to confirm your inquiries.

If you are familiar with the process of creating Web pages and are prepared to do at least a little reading, finish this chapter because it is important. It describes at a high level the challenges that Exclamation solves for Web site designers, developers, and content contributors. On the other hand, it is just background material. Feel free to skip to the next chapter at any point in your reading.

If you are a content contributor, once you have finished this chapter, move on directly to Chapter 6 “Editing content” and continue from there.

Coming soon, there will be a separate book for the content contributor (an added value for the designer to provide for clients).

Tools

Web designers use a wide array of tools for creating and maintaining their Web sites, but the most popular ones for the kind of size and sophistication we are talking about are Macromedia Dreamweaver and Adobe GoLive as well as “electric” HTML editors such as Bare Bones Software BBEedit and Macromedia HomeSite. Exclamation does not compete with these products (nor with the raft of image and graphic manipulation tools such as Jasc Software’s Paint Shop Pro, Adobe Photoshop or Illustrator. Rather it supplements them.

The tools which create pages on the fly tend to be expensive, and more importantly generally need to be located on the Web server. This in turn generates performance demands, and creates security challenges because all the sources are outside the firewall.

By contrast, Exclamation builds its pages on the desktop computer, and simple HTML pages are all that needs to be pushed to the Web production machine or hosting service. The pages are simple in terms of their lack of scripting or need for database calls, however Exclamation places no restrictions whatsoever on the complexity of the HTML pages—anything supported by your Web server is also supported (or safely ignored) by Exclamation.

In terms of the production tools, again, Exclamation does not compete: you can (and should) go on using any and all of the tools you are familiar with. This manual will point out when tool-specific behavior affects how you should use Exclamation.

Exclamation’s view of the world

A traditional Web site comprises one or more HTML files and their attendant graphic, audio and video files. Sophisticated Web sites may employ a tool such as JSP¹, ASP² or Cold Fusion³ for generating pages on the fly, and therefore allow the page to, for example, gather data from a database (sorted and filtered

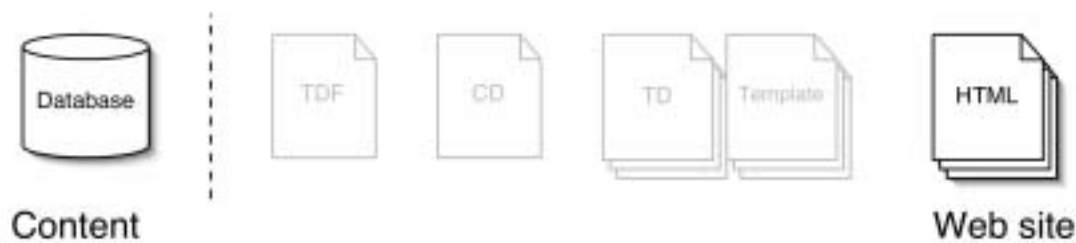


-
1. JSP™ is a trademark of Sun Microsystems
 2. ASP™ is a trademark of Microsoft Corporation
 3. Cold Fusion™ is a trademark of Macromedia Inc.

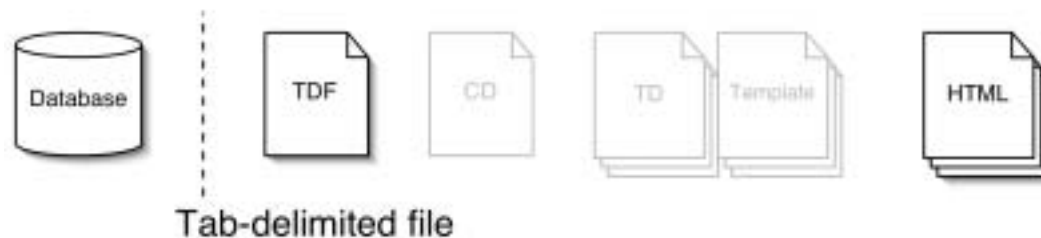


per a users instructions). A timetable page may only be populated after the user selects the route and travel date. These pages are generated from a database of information that the tools use to populate a predefined HTML table. Catalogs and other libraries of information are particularly suited to this treatment because of their repetitive nature (same format, different information on each page) but almost anything can be created this way.

Exclamation allows you to add the power and capability of a database at a fraction of the cost and complexity of traditional solutions. Let's take a quick run through how Exclamation achieves this. We start with your original HTML pages (or just the designs on paper) and we'll see how Exclamation is able to take the as-yet unconnected database containing the data you want to include, and use it to generate a page (or more likely a set of pages) that replicates the original page.



In order for Exclamation not to care about the details of the database, the data must be exported into a tab-delimited file. This is an absolutely standard, straight-forward practice. Even your Palm Pilot can export its data into such a file. In the illustration, we've separated the tab-delimited file (TDF) off from the



database with a dotted line because it is only the tab-delimited file that Exclamation cares about. In fact, it is perfectly possible for extremely simple

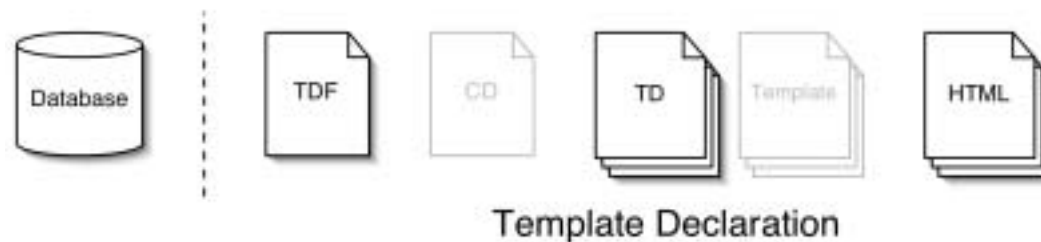
applications to have a hand-created tab-delimited file and not have a database behind it at all. Typically there will be one, or several tab-delimited files, but a dozen or more would not be unusual. We'll assume there is just one for now.

So now we have all the data in a file that Exclamation can read. The file consists of rows of data, with each row containing the data for one individual. This file has as many rows as there are people in the database. Each row might contain five pieces of data, say, first name, last name, home phone, cell phone, and e-mail address. These form the columns:

first	last	home	cell	e-mail
Snow	White	800 555 1234	888 123 9876	snow.white@enchantedforest.com
Fred	Flintstone	800 123 5678		fred@bedrock.net
	Shrek	800 321 9753	888 789 5678	shrek@pixarpics.net

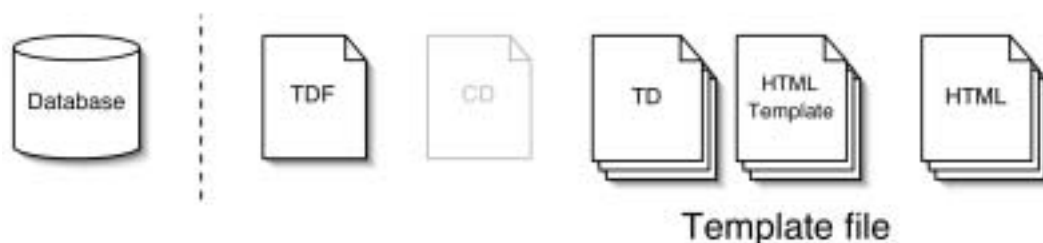
The columns are separated by tabs. Notice that it is okay for a column to be empty. The important thing is that the piece of data in each column always represents the same thing—all the first names are in the first column, last names in the second column, and so on. Exclamation needs to know how many columns there are, and to assign a name to each of them, so that they can be referenced.

Now here's the tricky part. The information is *always* extracted from the TDF file in the form of a query. "Go to the TDF file and pull out every record that is not blank in the e-mail column, then sort the list in alphabetical order by last name." The result is a list, where the rows are a subset of the rows in the TDF,



where the sublist comprises only those individuals that have an e-mail address. Exclamation calls this a *rowlist*. These instructions (or queries) are stored in Template Declaration files. Here again it is typical to have multiple Template Declaration files.

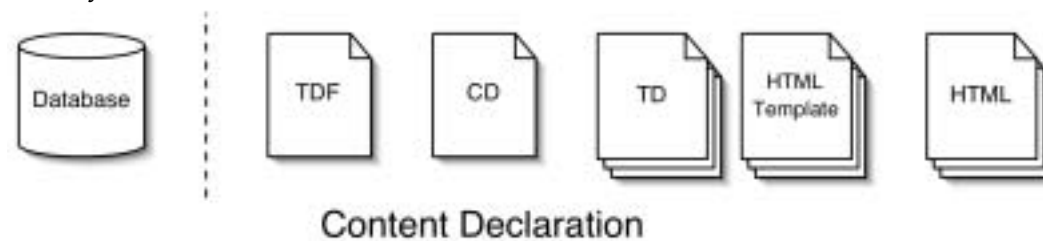
There's a direct connection between the Template Declaration and the HTML pages on your site, so let's take a moment to look at this relationship. Let's suppose you want a page that lists out all the names and phone numbers. That page will need a Template Declaration for its query. But suppose further you want to add a "detail" page for each person. This will also need a Template Declaration. Although a detail page for each person represents a large number of additional pages, all these extras are exactly the same format—the only difference between them is the content. So although there are a lot of pages, you'll only have two *types* of page: the long list page, and the detail page. Therefore you'll only have two Template Declarations. This is the critical point: you need a Template Declaration for each *type* of page, not for each *actual* page.



This Template Declaration file has a matching Template file. The Template is an HTML file. It provides the look and feel for the page. On the surface, the Template file looks like an ordinary HTML file, and in most respects it is. It can be created and edited in your favorite HTML editor. However, embedded in it are substitution expressions that will be replaced with real content when Exclamation creates the final HTML page.¹

1. In fact, if you have used Dreamweaver to create pages that are populated from, say, Microsoft Access tables, you'll be right at home with the Exclamation's HTML templates.

Finally, we need a file to tell Exclamation the name and location of the tab-



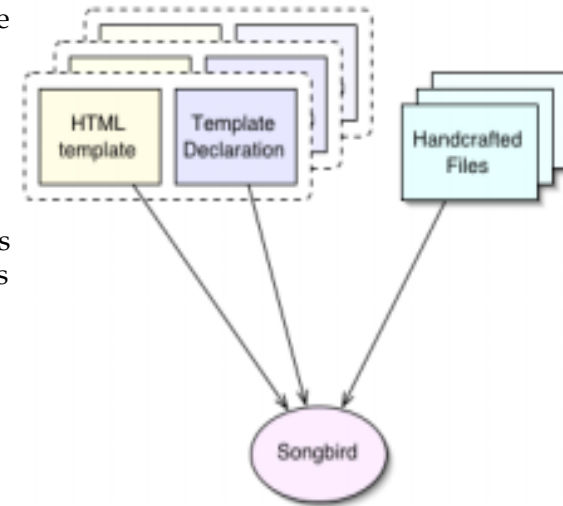
delimited file, and to describe each of the columns. This is called the Content Declaration (CD) file. There is only one for the entire project, because even if we had multiple tab-delimited files, they can all be described in the single Content Declaration file.

Exclamation uses the Content Declaration file to find the listed columns in the stated tab-delimited file, sorts and organizes the data according to the instructions in the Template Declaration file, and uses it to populate the HTML files it creates from the Templates. As part of the magic, if the Template Declaration calls for multiple pages (the detail page for each person for example) Exclamation takes care of that by creating as many pages as necessary.

Exclamation puts all the newly created HTML files in a Preview folder. If there are any HTML files that are complete and need no content—for instance you might have a hand-crafted “About Us” page—then Exclamation copies these into the Preview folder also. You can then use your favorite browser to view the newly created site.

Exclamation rebuilds relevant parts of the site each time the content is changed. This allows it to build any extra pages that the change necessitates, as well as updating any of the lists and tables in existing pages. It therefore splits the world into files that it needs to process (the Templates), and those that it does not (the hand-crafted files).

Regenerating files may seem tedious, but it has several distinct advantages. First, as noted earlier, it means that the pages are simple HTML that can be hosted on any Web server. Second, the design can be changed without re-entering the content. That makes the site vital, attractive and never stale. For



example, by maintaining several sets of template pages, (say a summer and a winter look) the site can be rebuilt with the new look and up-to-date content in a few seconds. The last, and most significant benefit is that it completely separates site design from content maintenance. And since regenerating the site is a matter of clicking on the buttons (without touching, and potentially breaking, the HTML code), the client can therefore update the site as frequently as she wishes.

This is a “50,000ft” view of how Exclamation works, and the files that it uses. This book guides you through the creation of each of these files in much greater detail. The SongBird Designer’s Reference Manual is structured with a chapter for each file type (Content Declaration, Template Declaration and Template) so this overview should help you find your way around that too.

Scenarios for building a production Web site

In the real world, there are two ways a Exclamation Web site is likely to be generated: from scratch; or from an existing Web site.

If you are building from scratch, it is worth considering putting as much as possible into the database, leaving as few hand-crafted pages as possible. This dramatically increases the client’s ability to maintain the site.

If you are converting from an existing site, we recommend the opposite approach. Convert the obvious candidates first, such as those with large numbers of repeated pages (same design, different content, such as the sample site’s movie detail pages), or the ones that change frequently, and get them running first.

A word about the examples used in this book

This book uses a sample database containing information about movies. There are three types of movie:

- those that are in the owner’s library
- wishlist items

- recent releases which look interesting, but which need to be viewed before a buy/delete decision is made.

All the examples in the book are based on this database. The database, along with all the files described and used (and the code snippets taken from them) are provided in your Exclamation installation so that you can examine and use them in any way you wish.

As you follow the examples in the book, you have a number of options:

- Obviously you can just follow by reading. We strongly recommend you take a more active role, and create a Web site for yourself, using one of the remaining options.
- Create your own examples by copying those in the book. In the early examples the whole file is described, so they contain everything you need. Later examples build on these original files, so the effort is not wasted.
- Use the files in the sample projects folder. This avoids the need to enter the code, but be aware that the files are *slightly* different from the examples. The book has stripped the HTML to bare bones, so that the focus is on the Exclamation code. The sample files function in *exactly* the same way, but the HTML contains a little more detail to pretty up the output. If you are at all familiar with HTML this should not be a problem.
- For the adventurous, try using your own material. For this you will need to create a tab-delimited content file, which you can generate from any number of applications, or even from scratch if it a small sample.

In the next chapter we'll use Exclamation to create a one page Web site consisting of a simple list of movies. Obviously this is not a realistic example, but it gives us a chance to discuss the basics of Exclamation operation without getting bogged down in the details. We will use this simple beginning and gradually add these features:

- select some movies from the database table
- link each movie title to a detail page with a full description of the movie
- break the long list down into separate lists by movie category

- move the category lists to their own files, adjusting the home page to merely list all the categories, plus an absolute link to a page listing all movies together (the original home page)
- on each category page, segment the movies by their audience rating (G, PG, PG-13, and so on). This is not as easy as it first appears, because we want the ratings in age order, not alphabetical, so we'll look at how to overcome this challenge.
- finally, in the advanced chapter, we'll look at some powerful additional features that Exclamation provides such as `vanish` which makes HTML tags or their content disappear if they are not needed.

Note that the examples are designed to build on one another in a gentle progression intended to ease the learning curve. It is *not* intended to demonstrate a practical way to deploy a real Web site. A more realistic approach to building a real site would be to mock up a site with a sample of each page, and once the mock-up gains design approval, replace each of the sample pages with a Template Declaration/Template pair. Test each new set of pages before moving on to the next one.

INTRODUCTION TO EXCLAMATION

A word about the examples used in this book

-
-
-
-

Creating a One-Page Web Site

As we started to discuss in the previous chapter, candidates for Exclamation-managed Web sites typically have some sort of catalog structure, but what's more important are two key features:

- The content is required to change on a regular basis. This may be a simple additive process—today's news stories, this week's reviews, this month's releases, or things can require editing and deleting—inventories, scoreboards, and so on.
- The content is required to be spread over a number of separate pages which maintain a consistent style and format (or a family of styles and formats)

Exclamation Web Sites

A Exclamation Web site comprises the following:

- A Template for each of the basic page types required on the site. There can be as many as you like, arranged in any kind of folder structure.
- For each Template there is a Template Declaration file, which tells Exclamation where the content for the page is stored, and how to access it.
- A Content Declaration file listing all the content sources (the tab-delimited files). It provides tables describing the format of the data in those files.
- One or more content files. These files are used by Exclamation to populate the templated pages. There are many forms and formats the content files can take, but a typical example is a tab-delimited file exported from a desktop database or spreadsheet application.
- As many hand-crafted pages as you want. The content and format of these pages will continue to be maintained outside Exclamation.



- A project file which tells Exclamation how all the pieces fit together, and where they are located.

These files are all arranged in a simple folder hierarchy that form the project's Site Design. Exclamation uses the project file to build out the relevant pages and copies the hand-crafted (non-templated) material into the relevant directories also.

Once the designer is happy with the Web site, she provides the content contributor with the database that the client can use it to keep the content up-to-date. All the content contributor needs to do is apply a change the database, and export the data to the tab-delimited file. Finally the client uses a simplified Exclamation to rebuild the site.

Thus once up and running, clients can make whatever *content* changes they wish, and update the site as often as they wish without fear of damaging the style or organization. If extra content pages are required, Exclamation takes care of that. This frees up the designer from much of the site administration and allows them to concentrate once more on the design.

There are a couple of other advantages to Exclamation's approach. It makes it trivial to put the same information on multiple pages, or to put different pieces of information about the same item on multiple pages. Finally, the content contributor does not need an intimate knowledge of the Web site in order to know where changes need to be applied—all changes can be made in the database, and Exclamation will propagate them as necessary.

A word about Dreamweaver and GoLive Templates

At first glance Exclamation templates appear to perform a similar function to Dreamweaver and GoLive templates, but in fact they are very different and should not be confused.

Dreamweaver and GoLive create a collection of hand-crafted pages. These templates make it easy to update a style and also provide a mechanism to prevent content contributors from changing anything that is controlled by the template. However the templates do not manage the content. If the same content appears on multiple pages, changing the text means editing all the pages it appears on. If you add a page (for example you add a new movie to

the list) the page is created by hand using the template as a guide, and then the content is added. If you decide to add a field (say the movie's release date) the template can update the pages to include the field, but then the content must be added by hand on each of dozens or thousands of existing pages.

By contrast, Exclamation templates control both the appearance and the *content* of the page. As each page is built, the relevant content is pulled in from data sources. This is a critical difference. If you want to add the movie's release date to the page describing the movie this is a trivial task in Exclamation. Exclamation automatically updates all instances of the content, no matter where, or how often, it is used throughout the site (making it especially easy to maintain multiple versions of the site).

Another critical advantage is deletion. Site creation and maintenance is typically an additive process: the tools do not know which pages are obsolete and should be deleted and so the production area steadily accumulates "dead" files. Because Exclamation builds the site from scratch it knows that the "staging area" (Exclamation's Preview folder) is accurate and any files which are deleted from there Exclamation automatically clears from the production site during the upload.

Dreamweaver and GoLive use multiple templates within a site to achieve different looks for different pages (for example different sections of the site might have a different look). Exclamation can do that too, but in addition different templates can also be used to completely change the look of the *same* pages (a summer and a winter catalog for example).

Finally, because Exclamation knows how many pages to build (depending, for example, on the number of movies currently in the database) it also automatically builds the page-to-page links: a fast and reliable way to ensure navigational integrity.

Exclamation site structure

Although Exclamation does not place restrictions on the structure of the development site, a consistent structure is recommended. The following structure is used throughout the documentation. The top-level Exclamation project folder contains three subfolders:

- Content
- Site Design
- Preview

In addition Exclamation maintains the Web site on the production server.

Content

The content folder contains

- The database (one or more FileMaker, Access, Excel, or other application tables or spreadsheets). All that is required of these source applications is that they can export the data into a tab-delimited file.
- The tab-delimited files.
- A Content Declaration file listing all the content sources (the tab-delimited files). It provides tables describing the format of the data in those files. Each table maps one-to-one with a tab-delimited file.

Site Design

The Site Design folder contains the Templates and hand-crafted files that dictate the look and feel of the Web site. The difference between this set of files and the final pages to be built in the Preview folder is that the Template files contain references to the content.

Exclamation allows you to maintain multiple site designs using the same content database. (We'll use this feature through this book—each chapter's examples are saved as a separate site design.) So the Site Design folder can contain multiple sub-folders, each representing a single site design:

Site Design (folder with fixed name)

design-name-1 (site design folder)

design-name-1 (Template Tree—folder containing actual Templates and hand-crafted files)

[other folders & files]

design-name-2

design-name-3

Note that the individual site design folders *must* contain a sub-folder with the same name, and that this sub-folder is what actually contains the Templates and hand-crafted files. This allows Exclamation to separate out the “meta” files and folders that may need to be associated with the site, but which are not actually part of it. Example meta files might be a GoLive’s design-name-1.site file and design-name-1.data folder, a BBEdit “file group” file, or Photoshop source files.

Any folder in the Site Design folder is considered to be a site design folder and its name is used by Exclamation to populate Exclamation’s Site Design pulldown list.

**Preview**

The preview folder is where Exclamation constructs the entire Web site, merging generated and hand-crafted files. The Preview folder follows the structure you created for the Site Design folder. So each site design you create is in its own folder.

You can use this preview “site” to review and test the pages before releasing it to the public.

Creating a Web site

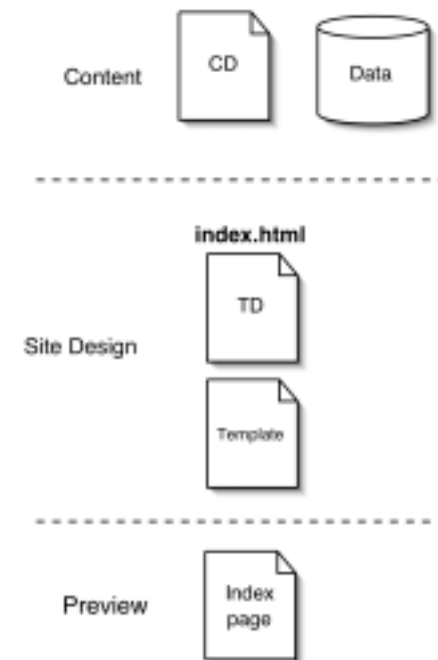
At a very high level, these are the steps:

- 1 Typically the first step is to design mock-ups, create scenarios, and lay out the basic structure. These are frequently the materials presented to the client for sign off, and because it is so much easier to change the mock ups than the actual sources, work does not generally start on the site proper until this has happened.
- 2 The first Exclamation step is to create the development site folder structure. Start with a project folder (in our case it is called *Movies*) and under it the three subfolders *Content*, *Preview* and *Site Design*.

- 3 Place the database in the `Content` folder. Export one or more tab-delimited text files of the content.
- 4 In the same folder create a Content Declaration file, which tells Exclamation where to find the content file, what the columns (or fields) are named, and what type of information is in each column.
- 5 In the `Site Design` folder, create the HTML Template page(s). These are standard HTML pages except that the content which is to be pulled from the database is missing, and in its stead are substitution instructions.
- 6 In the same folder, create the Template Declaration file(s). This contains the database query and also a declaration statement instructing Exclamation on the name of the Template file, where to put the output, the name and location of each generated file, and so on.
- 7 Also in this folder, create any hand-crafted files. These include graphic files and style sheets as well as HTML files that are not affected by Exclamation. In short, this folder should contain any and all files needed by the Web site.
- 8 Open Exclamation and create a project that ties everything together: explaining the directory structure you created, where you put the declaration files, and what you called them.
- 9 Use the Exclamation to render the template file. Exclamation places the output file in the `Preview` folder.

Let us start by creating the simplest possible site: a one-page listing. We will assume that there is already a database, and that it consists of one table, called `movieList`, which has been exported from its parent application, and is now sitting in our content directory ready to be used.

The first two steps (creating the Template and its Template Declaration file) could be carried out in either order. Indeed it is common to switch back and forth between them. For example, you might start by creating the look and feel for the page (you may even be starting from an existing page). Then work on the Template Declaration to calculate the query. After testing and working out any bugs in the query, you could then return to the Template to fine-tune the layout now that you can see what it looks like when there is real data in the page.



The sample files used in this section are based on those in the “Site Design/onepage” folder. The Content Declaration file (located in the Content Folder) is described first. It is used (unchanged) by all the sample Web sites we build in this book.

Creating the Content Declaration file

The Content Declaration is a simple text file. You can use an editor of your choice (for example Dreamweaver, GoLive, or BBEdit) you just need to be able to edit the source code.¹ Here’s the `movies.excd` file we created for `movies`.

Movies Project/Content/movies.excd

```

1  <?xml version="1.0" ?>
2  <!DOCTYPE content SYSTEM "jar:/Content.dtd">
3  <content>
4      <tabdatabase>
5          <table name="movies" file="movies.txt">
6              <column name="id" type="integer"/>
7              <column name="Title" />
8              <column name="MPAA" from="11" />
9              <column name="Rating" from="19" />
10             </table>
11         </tabdatabase>
12 </content>

```

XML requires the first two lines.

The `table` name attribute is needed in order to refer to this table. In this case it allows a Template Declaration file to find the table containing the relevant content for that template.

For each column that we want to use from the source data’s csv file we need a column tag with a named data type. `Safetext` is the default, but we’ve changed the `id` to `integer` so that its numbers will sort correctly (for example 10 will appear before 3 if numbers are treated as any kind of text).

These are not the next two columns in the tab-delimited file. We can skip the columns we don’t need by using the `from` attribute to identify the column number we want to use. (Columns are numbered from left to right in the file.)

The first (and only) table closes, the table section closes, and the content declaration is complete.

Note that for clarity, many of the columns we’ll need later are not listed here, but can be found in the actual sample file.

1. The Macintosh OS X TextEditor is not suitable, despite its name, because it creates rich text files.

Creating a Template Declaration File

Exclamation uses pairs of Template Declaration and Template files to create the HTML. First we'll create the Template Declaration file. Again you can use your choice for an editor, you just need to be able to edit the source code. The declaration performs two important duties:

- It tells Exclamation the name of the Template file, what to call the "generated" (output) file (and where to put it) and the format to expect from the tab-delimited file.
- It provides the query string (or strings).

This file uses XML. There is a brief discussion of XML in the Reference Manual. The file is called `index.extd`, and can be found in the `Site Design/1 - One Page` folder.

Movies Project/Site Design/1 - One Page/index.extd

```

1  <?xml version="1.0"?>
2  <!DOCTYPE template SYSTEM "jar:/Template.dtd">
3  <template>

4  <!-- One Page of Movie Titles -->
5  <input file="index.html"/>
6  <output file="index.html"/>

7  <!-- List of Movies -->
8  <query table="movies" sortBy="Title">

9  <rowlist name="movierow" />
```

XML requires the first two lines.

The declaration statement.

Standard HTML comment.

The declaration provides the name of the Template file to be used (which we will create next), and tells Exclamation to create an output file called `index.html`. Since the Template Declaration file is named after the input filename (`index.html`) the input statement is not required, but is shown here for clarity.

Another comment.

The query. Notice the `table` attribute, which has as a value the name of the table in the content declaration file. The second attribute, `sortBy` causes the query result to be listed in `Title` order, where `Title` is the name of the column in the table (line 12 in the Content Declaration file above).

The query is going to produce a list of all the movies that satisfy the query (which is all of them at the moment), and we're going to need to refer to this list, so we've called it `movierow`.

```
10 </query>
11 </template>
```

End query statement.

Filtering content using Keep and Omit

Databases may contain records that should not be published in the Web site. For example, in the sample database there are movies that are not actually in the movie library (they are on a wishlist, or have not even been seen yet). Exclamation allows you to filter the list using two condition statements: `keep` and `omit`.

Keep

`Keep` is extremely useful when you are using “dirty” databases, that is, databases containing records that are not needed for this Web site. We need to use a `<keep>` statement in our query to select just the movies that are in the library.

Movies Project/Site Design/2 - Simple/index.extd

```
1 <query table="movies" sortby="Title">
2 <keep> movies.OwnStatus EQ "Own" </keep>

3 <rowlist name="movierow" />

4 </query>
```

The query.

Keep only those records which have the value “Own” in the `OwnStatus` column. So we’re going to eliminate all the “Wish List” and “Need to see” movies. Now our rowlist will only list the movies that are actually in the library.

Note that the `keep` statement must refer to the query table, and not the rowlist, because the list does not actually exist yet. The `keep` will already have done its job when the rowlist is created (so it will already be filtered to contain just the “owned” movies).

End query statement.

If you have more than one `<keep>`, they are logically-OR'ed together. That is, a row is a keeper if any one of the `<keep>` expressions is true. This may not give you the results you expect. For example, maybe you want to list only those movies that you own *and* are part of the given category. If you do this:

```
<keep> movies.Category EQ "Comedy" </keep>
<keep> movies.OwnStatus EQ "Own" </keep>
<rowlist name="movierow"/>
```

The first `<keep>` requests that the resulting list will include a movie if its Category matches a specified Category name ("Comedy"). Because the `<keep>`s are OR'ed, the second `<keep>` adds any movie that has the value "Own" in the OwnStatus column. You get both. What we really want to do is list all the movies that have the value "Own" in the OwnStatus column, but only if they *also* have a Category with the value "Comedy." To achieve this, use a single `<keep>` with an AND:

```
<keep> movies.Category EQ "Comedy" AND movies.OwnStatus EQ "Own" </keep>
```

As things get more complicated, use parentheses to ensure that the OR logic is resolved first:

```
<keep> (movies.Category EQ "Comedy" OR movies.SubCategory EQ "Comedy")
AND movies.OwnStatus EQ "Own" </keep>
```

Omit

There is one record in the sample database that is not a movie. It lists the column name in each column. For example the Title column has the value "Title" and the Category column has the value "Category."¹ Here's how we can eliminate this record from our query:

```
1 <declare> Business as usual
2 <input file="index.html"/>
3 <output file="index.html"/>
4
5 <!-- List of Movies -->
6 <query table="movies" sortBy="Genre, Title">
7 <keep> (movies.Category EQ "Comedy" OR movies.SubCategory EQ
  "Comedy") AND movies.OwnStatus EQ "Own" </keep>
```

1. This turns out to be useful when importing and exporting data from the database. It is not relevant, and it could have been removed, but it provides us with an easy target for demonstrating `<omit>`.

```
8 <omit> movies.Category EQ "CATEGORY" </omit>
```

```
9 <rowlist name="movierow" />
```

```
10 </query>
```

```
11
```

```
12 </declare>
```

The omit statement. Again, the omit refers to the query table, because the rowlist does not exist yet.

A row is put in the row list if it matches *any* <keep> and *none* of the <omit>s. You can have as many keep and omit statements as you like. If your database consists of records of type "a" and records of type "b" then it is clear that keeping records of type "a" has the same effect as omitting records of type "b" and so either could be used.

We'll save this file as `index.html.extd` (which matches with the `index.html` HTML template name). By doing so we make it easy to tell which Template Declaration goes with which HTML template, and we eliminate the need for the `input` statement at line 2 above).

Creating the Template file

Next we need to create the (HTML) Template file. This is done using your favorite HTML editor. Exclamation makes no special demands on HTML creation, so once again any editor will do. The file is stored in the Site Design directory.

Here's the sample file. In the interests of clarity, the file contains no extraneous code, but your file can be as complex, colorful or graphic-laden as you wish.

Only lines 15 through 19 are important to Exclamation:

Movies Project/Site Design/I - One Page/index.html

<pre> 1 <html> 2 3 <head> 4 <meta http-equiv="content-type" content="text/html; charset=ISO- 8859-1"> 5 <title>All Movies by Title with Reviews</title> 6 <link rel="stylesheet" href="default.css"> 7 </head> 8 <body bgcolor="#ffffff"> 9 10 <h1>All Movies by Title</h1> 11 12 <p>This is a simple list of every movie in the database: those in the collection; those we'd like in the collection; and those we'd like to see in the theatres/rent first. Eventually, you'll be able to click on a movie title from the list below and see a page with a lot more detail on the movie.</p> 13 <h2>Movie List</h2> 14 <table border="0" cellpadding="2" cellspacing="0" width="750" height="18"> 15 <tr valign="top" LOOP=movieRow> 16 <td> <p>[[movieRow.Title]]</p> </td> 17 <td> <p>[[movieRow.Category]]</p> </td> 18 <td> <p>[[movieRow.MPAA]]</p> </td> 19 <td> <p>[[movieRow.MyRating]]</p> </td> </pre>	<p>Absolutely standard HTML header information.</p> <p>Start of body.</p> <p>More standard HTML</p> <p>Beginning of the table which will contain all the magic.</p> <p>This is a Exclamation loop statement. Because it is inside an HTML <code><tr></code> tag, it will end at the <code></tr></code> tag. If there were no loop statement, the displayed page would only contain one row. This simple loop statement says "now loop over each row of the movieRow list."</p> <p>The loop starts at the beginning, and continues until it reaches the last row. Thus the page will display one row for every row in the movieRow list.</p> <p>These four cells contain Exclamation substitution instructions instead of text. The first says "for the current row in the movieList query, insert the content of the Title column."</p> <p>The second says "from the same row, insert the content of the Category column."</p> <p>The third and fourth say the same thing for the MPAA and MyRating columns.</p>
---	--

```

20 </tr>
21 </table>
22 <p>This page was generated by the Exclamation Desktop Publisher
    from Ten League Boots.</p>
23 </body>
24 </html>

```

After we've been through the entire publishing process once, we'll return to this file and expand its capabilities. For now, we'll save it as `index.html` (which is the name the Template Declaration file is expecting in its input statement).

The table row ends, and therefore so does the loop, so that in the output file the row will repeat.


The table closes.

The end body and end html that provide the standard close for HTML pages.

Using Exclamation to create the project

Now we have all the elements we need to create the actual Web page. The next thing we need to do is to create a Exclamation project, which will tie all these files together.

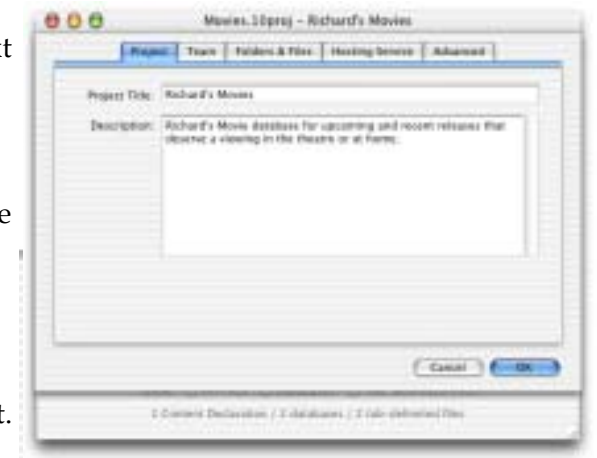
- 1 Launch the Exclamation application.
- 2 Choose File-> New Project. The project Options dialog page opens with the Project tab selected. Choose a project title, and if you wish, fill in a description.
- 3 The Folders and Files tab is the important one. Its four fields need to be filled in by browsing to the relevant folder:
 - Project Folder: the full path name to the folder you created for the project. In our case it is:

 /Users/yourname/Exclamation/Movies.

:

- Content Decl: the location of the Content Declaration file. As the example shows, this is a relative path from the one provided above for the project. We put the content declaration file in the content folder: `Content/movies.excd`

 Click the Select button to browse for the file.






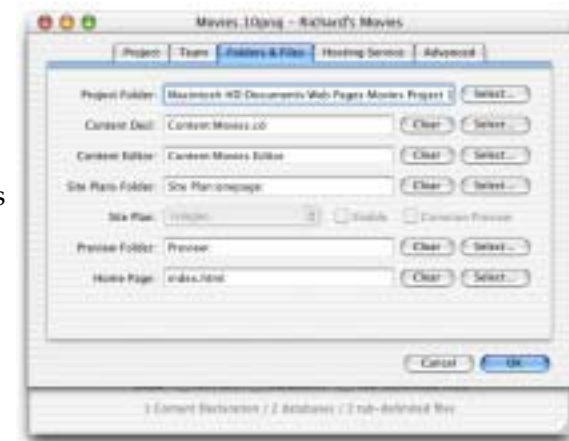
- Content Editor:
 - Site Design Folder: this is the folder we created in which we developed the template file: `Site Design`.
 - Site Design: If you have more than one Site Design, this pull-down allows you to easily switch between them. If you check the Common Preview box all the Site Designs will build in the same Preview folder. Uncheck the box to use separate Preview folders.
 - Preview Folder: as we discussed earlier, this is where Exclamation will build your site. It is currently empty, but we did create the folder, so we can add the folder name: `Preview`.
 - Home Page: If your home page is not named `index.html` then click on Select to browse to the file you are using. Exclamation will use this as the file to open when you click on preview your results.
- 4 Click on OK to dismiss the dialog.
 - 5 Choose File -> Save to save the project file. Browse to the Project folder and save the file there. Our file is called `movies`. Exclamation automatically adds the extension `exproject`, so the file name (which is reflected in the title bar of the Exclamation window) is `movies.exproject`. The project title is the one we gave it in the Project Options dialog.

Now the project is set up, we are ready to generate the files we need.

Rendering the Template

- 1  Click on the Make all Pages. You are done!
- 2 Exclamation will automatically open the Preview list. (If there are errors, it will also open a Message Log page.)
- 3 Click on the Preview Pages button. Your default browser will open and display the new page.

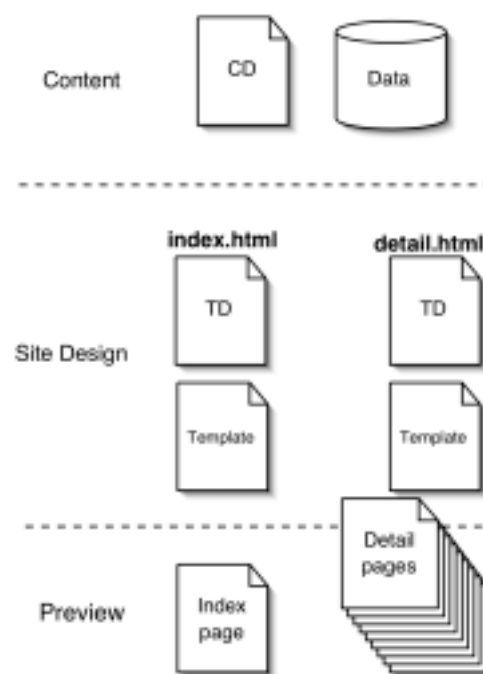
There are other options and features in Exclamation that we'll cover later (when we have more files to deal with). But for now, we're done: we've rendered our first Web site using Exclamation!



Multiple Output Files

Now that the movies in our database are displaying as a list on our index page, the next step is to create a detail page for each movie. The goal is to be able to click on the title of the movie and display a new page with further information about the specific movie.

We only need one more Template page, which will describe the look and feel of all detail pages. Exclamation will use the Template to generate as many detail pages as there are titles in the movie database. But first we need to make one slight change to the original Template file which will add the links to the new detail pages. Neither the Content Declaration nor the Template Declaration need to change. The sample files used in this section are based on those in the “Site Design/simple” folder.



Creating a Template with multiple output files

If we were to manually create all the pages, a typical link in our `index.html` file might look like this:

```
<td><a href="starwars.html">Star Wars</a></td>
```





We know from our experience creating the index page that Exclamation allows us to create a list of movies on the page by using this code instead:

```
<td><a href="starwars.html">[[movielist.Title]]</a></td>
```

With this code, the movie titles will generate correctly down the list, but the example will currently generate the same link to the same (`starwars.html`) file for every movie. But we can add code to the anchor also:

```
<td><a href="detail-[[movieQuery.id]].html">[[movieQuery.Title]]</a></td>
```

If the ID for the movie Star Wars is 123, this line will appear in the rendered file as:

```
<td><a href="detail-123.html">Star Wars</a></td>
```

Compare the table in the original `index.html` with the new one:

	Site Design/1 - One Page/	Movies Project/Site Design/2 - Simple/Index.html	
1	<code><table></code>	<code><table></code>	No change.
2	<code><tr LOOP=movierow></code>	<code><tr LOOP=movierow></code>	No change.
3	<code><td></td></code>	<code><td></td></code>	

4	<pre><td><p> [[movierow.Title]] </p></td></pre>	<pre><td><p> [[movierow.Title]] </p></td></pre>	<p>href added to <code>movierow.Title</code>. Note that the href contains a substitution, so that in each row of the generated table, the link will be different (it will be the link required to go to the detail page for the named movie).</p>
5	<pre><td><p> [[movierow.Category]] </p></td></pre>	<pre><td><p>[[movierow.Category]]</p></td></pre>	<p>No change.</p>
6	<pre>[[movierow.MPAA]] </p></td></pre>	<pre><td><p>[[movierow.MPAA]]</p></td></pre>	
7	<pre>[[movierow.Rating]] </p></td></pre>	<pre><td><p>[[movierow.MyRating]]</p></td></pre>	
8	<pre></table></pre>	<pre></table></pre>	

We've merely added an href anchor (link) to the existing code inside the table cell tag. In the same way that `movieQuery.Title` will be replaced with the title of the movie, `movieQuery.id` will be replaced with the `id` (a number) for that movie.¹ So each row in our movie list will now contain the name of the movie, and it will be a link to a file whose name matches the `id` of the movie, for example `detail-123.html`.

Creating the new Template file

As before, use your favorite editor to create the new Template file. This file (`Site Design/2 - Simple/detail.html`) defines the layout for the

1. We could have used the movie title again, with the advantage that the file names would make sense, but we also run the risk of the names not being unique and of files being created with spaces in their names, which does not sit well with some servers.

detail page describing each movie. The file contains no surprises: it uses the same syntax as we used in the previous Template file.

Site Design/2 - Simple/detail.html

9	<code><body></code>	Start of body.
10		
11	<code><h2>[[movierow.Title]]</h2></code>	
12	<code><table border="0" cellspacing="0" cellpadding="2"></code>	More standard HTML
13	<code><table></code>	Beginning of the table which will contain all the magic. Note that there is no <code>loop</code> statement: this table is a set format that will have its content provided by the information in the current row of the <code>movieQuery</code> query result.
14	<code><tr></code>	
15	<code><td>[[movierow.Category]]</td></code>	These three cells contain Exclamation database calls instead of text. The first says "for the current row in <code>movieQuery</code> , write the content of the cell in the <code>Category</code> column."
16	<code><td colspan="2">[[movierow.MPAA]] : [[movierow.RatingReason]]</td></code>	
17	<code><td >[[movierow.MyRating]]</td></code>	The second and third say the same thing for the <code>MPAA</code> , <code>RatingReason</code> and <code>MyRating</code> columns.
18	<code></tr></code>	
19	<code><tr></code>	
20	<code><td>Director:</td></code>	This row is slightly different, but the technique is the same: here the first cell contains fixed text (" <code>Director:</code> ") and the second cell contains the call for the director name for this row in the query result. Then the same for " <code>Cast</code> ".
21	<code><td>[[movierow.Director]]</td></code>	
22	<code><td>Cast:</td></code>	
23	<code><td>[[movierow.Cast]]</td></code>	
24	<code></tr></code>	
25	<code><tr></code>	
26	<code><td colspan="4">[[movierow.Overview]]</td></code>	Then comes a final row for the movie description.
27	<code></tr></code>	
28	<code></table></code>	The table ends...
29	<code><p>Back to Home</p></code>	
30	<code></body></code>	...and so does the file.
31		
32	<code></html></code>	

Creating the second Template Declaration file


Given how similar our two HTML files are, it should be no surprise that the two Template Declaration files are also similar.

2 - Simple/index.extd	2 - Simple/detail.extd	
1 <?xml version="1.0"?>	<?xml version="1.0"?>	XML requires the first two lines.
2 <!DOCTYPE template SYSTEM "jar:/Template.dtd">	<!DOCTYPE template SYSTEM "jar:/Template.dtd">	No difference.
3 <template>	<template>	
4 <!-- 1 Page of Movie Titles-->	<!-- Detail Page for each Movie -->	Standard HTML comment.
5 <output file="index.html"/>	<output file="detail-[movierow.id].html" />	The output file declaration tells Exclamation to create an output file for each row in the rowlist. Each file will substitute [[movierow.id]] for the number representing the id of its movie, for example detail-15.html
6 <!-- Movie List -->	<!-- List of Movies -->	
7 <query table="movies" sortby="Title">	<query table="movies" sortby="Title, Category">	The query. The only difference being that we don't need to sort the output this time.
8 <keep> movies.OwnStatus EQ "Own" </keep>	<keep> movies.OwnStatus EQ "Own" </keep>	
<rowlist name="movierow" />	<rowlist name="movierow" />	
9		
10	</query>	End second query.
11 </template>	</template>	

Notice in particular line 6: the output filename is a substitution expression that will be replaced when we render the pages, and will generate a series of files called detail-1.html, detail-2.html, ..., detail-101.html, ... and so on.

Rendering the new pages

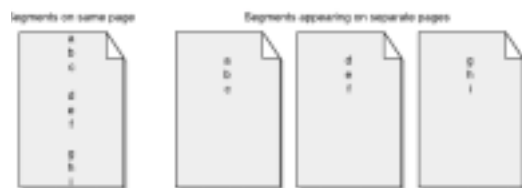
Now we are ready to test the new pages. If you are following along using the sample files, check that Exclamation is configured to use the correct project pages (verify that the Site Design pull-down menu has the Simple site selected).

- 1  Click on Make all Pages. Notice that this time many more files are created. These are all the new detail pages (one for each movie in the list).
- 2 Exclamation will automatically open the Preview list. (If there are errors, it will also open a Message Log page.)
- 3 Click on the Preview Pages button. Your default browser will open and display the new index page. Notice that each of the movies in the list is now a link.
- 4 Click on the link to display the detail page we just created.



Segmenting

The goal for this chapter is to take our original movie list and divide it into separate lists for each category of movie. This division process is called segmenting. First we'll simply adapt the original index page so that the list appears segmented into separate tables. Then we'll redo the page completely, so that each category listing appears on its own page—which of course Exclamation will generate for us.



The first stage requires no extra files. The second introduces category pages. We'll also save the original index.html page as movie-list.html so that the site still contains a simple listing of all the movies in the library.

Creating a simple segmented list

Up until now, our movie list has been generated as one long table in alphabetical order, by title. Now we're going to break it up into a series of tables, with one table for each category. The sample files used in this section are based on those in the "Site Design/Segmented" folder.

As before, we could create each table by hand, but now we would not only need to know exactly how many movies are in each category, but also how many categories there are. Segmenting will allow us to add (or delete) movies *and categories of movies* and recreate the page again at the push of a button. Like the movie detail pages in the previous chapter, the real power of Exclamation's generation technique is that you don't need to care how many segments there are, nor how many are added or deleted between builds. For example, at the





2002 Academy Awards, a new film category was announced: animation. We can go back into the movie database, add the category, and change the category for the existing animation features to the new category. The next time the site is built, a page segmented by category will include a table for animation.

Note that there are a number of fields in our database that we could use to helpfully segment the listing:

- Personal “star rating”
- MPAA rating
- Categories

For this example, we’ll use the star ratings. The other two fields present some interesting challenges which we’ll look at in the next chapter.

This example shows the first three segments in the new layout. You will also notice that the now redundant rating column has been removed. Each table is preceded by a heading, which is the rating. So whereas our original list of movies was a simple loop, segmenting uses two loops, one inside the other. In this case, the outside loop needs to contain the heading, and the beginning and



end of the table. Then the inner, row loop, contains the row that will repeat for each movie in the rating.

```
*****
Big Sleep, The      1946      Film-Noir      Approved
Bonnie and Clyde   1967      Crime          M
Casablanca         1942      Drama          PG
Chinatown          1974      Crime          R

****

American Beauty    1999      Drama          R
As Good As It Gets 1997      Comedy         PG-13
Boyz n the City    1980      Action         R
Chariots of Fire   1981      Drama          PG
Chicken Run        2000      Animation      G

***

18 Days            2000      Comedy         PG-13
3000 Miles to Grandd 2001      Action         R
39 Steps, The      1935      Mystery        Unrated
```

The example shows four movies in the 5-star segment, five in 4-star, and so on. So let's see how this was done.

Creating a new Template Declaration File

We need to create a Template Declaration (Site Design/3 - Segmented/index.extd) that describes the segmentation we want for the query. No surprise that it is going to look very like our previous Template Declaration files.

3 - Segmented/index.extd

```
1 <?xml version="1.0"?>
2 <!DOCTYPE template SYSTEM "jar://Template.dtd">
```

Absolutely standard HTML header information.

```

3 <template>
4
5 <!-- One page of Movie Titles -->
6
7 <output file="index.html" />
8
9 <!-- List of Movies segmented by Star Rating-->
10 <query table="movies" sortby="MyNum descending, Title">
11
12 <keep> movies.OwnStatus EQ "Own" </keep>
13 <segmentlist name="ratings" byequal="MyNum">
14
15 <rowlist name="movierow" />
16 </segmentlist>
17 </query>
18 </template>

```

No change here.

Start of query. Again no change.

So here's the change. The simple query was empty, now we've added the segmentlist tag (lines 13 and 15).

The `byEqual` attribute says "use the `MyNum` column to segment the list." So the list will be divided into separate sections for "5-Star" "4-Star" and so on.

Notice that the `sortBy` in the query at line 10 and the `byEqual` in the `segmentlist` are both sorting by the same category. This is required.

Rowlist as before, with a new name.

All's well that ends.

Adding the new Template file

As before, use your favorite editor to create the Template file (`Site Design/3 - Segmented/index.html`). The file contains the now familiar syntax for all our Template files. This time however, we need two loop statements for the segmenting: the outer loop generates a header and a table for each star-rating: the inner loop, as before, generates a list of movies. This time the inner loop works multiple times—once for each iteration of the outer loop.

3 - Segmented/index.html

```

1 <tlb LOOP=ratings>

```

The first (outer) loop opens. The outer loop is over the list of star ratings,

```

2 <H2>[[ratings.MyRating]]</H2>
3
4 <table border="0" cellpadding="2" cellspacing="0">

5 <tr valign="top" loop=movierow>

6 <td>
  <p>
    <a href="detail-[[movierow.id]].html">[[movierow.Title]]</a>
  </p>
</td>

7 <td><p>[[movierow.Copyright]]</p></td>
8 <td><p>[[movierow.Category]]</p></td>
9 <td><p>[[movierow.MPAA]]</p></td>
10 </tr>


11 </table>
12 <tlbend>

```

For simplicity, this example does not include the entire file, just the section we are changing since the previous example.

Rendering the new pages

Now we are again ready to test the new pages. If you are following along using the sample files, check that Exclamation is configured to use the correct project pages (verify that the Site Design pull-down menu has the Simple site selected).

- 1  Click on Make all Pages. Notice that this time many more files are created. These are all the new detail pages (one for each movie in the list).
- 2 Exclamation will automatically open the Preview list. (If there are errors, it will also open a Message Log page.)

As expected, the first loop contains a header and the opening of the table.

Notice that the header is itself a Exclamation statement, which will be replaced by the current star rating each time the header is looped over. The reference is to the `segmentlist` (which is named `ratings`).

The inner loop starts. This one refers to the `rowlist` which is named `movierow`.

The inner loop is over the titles in each star rating.

The row closes, which automatically closes the inner loop also.



- 3 Click on the Preview Pages button. Your default browser will open and display the new index page. Notice that each of the movies in the list is now a link
- 4 Click on the link to display the detail page we just created.

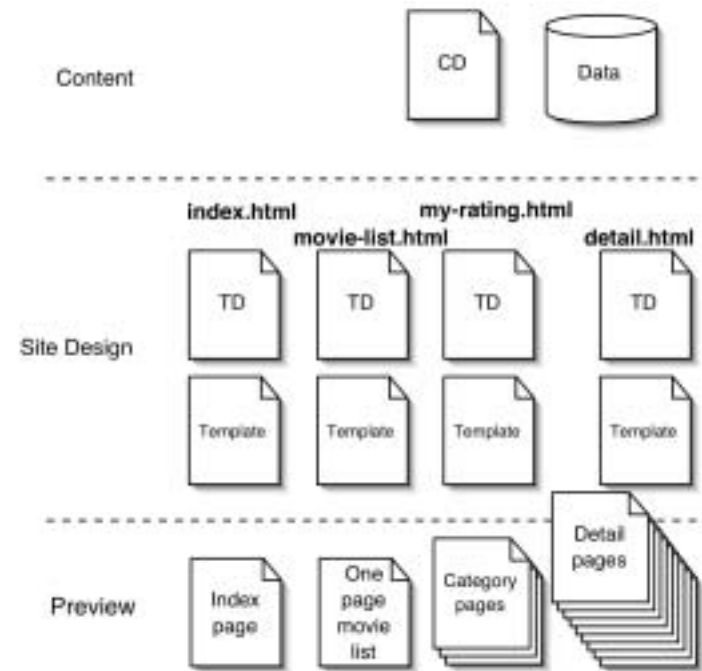
A segment with multiple output files

We've already seen how to use an Exclamation statement to automatically generate new files, so we can use the same technique for our rating files. The sample files used in this section are based on those in the "Site Design/4 - Complete" folder.

First we will redo the index page. The new format will consist of a simple list of the star ratings currently used in the database, which will be linked to new pages, each of which will list just the movies with that rating. A last page will list all the movies. The rating list pages are going to be much like the original rating page, except that they only list the movies for a particular rating. As before, users should be able to click on the title of a movie in the list to display a detail page. We can reuse the detail page from our second example. So now our project will contain four template files:

- `index.html` which displays a list of all the current star ratings
- `my-rating.html` which, for each star rating, displays a list of movies belonging in that rating.
- `detail.html` which displays the details for a particular movie.
- `movie-list.html` which displays the complete (simple) list of movies.

We'll need four declaration files to match these templates. The template for the detail pages is in fact the same as we've always used, so we do not need to change anything here. For the last template, the movie listing, we can just rename our original simple index page.



Creating the index Template Declaration file

Actually, although this is a different structure, the new home page is not dissimilar to the previous one: it is still a list of star-ratings, but this time it is just a simple list of ratings. The declaration file can therefore remain the same as before.

To show we're not using the `rowlist` expression we simply changed the name of the rowlist:

4 - Complete/index.extd

```

1 <?xml version="1.0"?>
2 <!DOCTYPE template SYSTEM "jar:/Template.dtd">
3 <template>
4
5 <!-- One Page of Star Rating segments -->
6
7 <output file="index.html" />
8
9 <!-- List of Star Rating segments -->
```

```

10 <query table="movies" sortBy="MyNum descending, Title">
11 <keep> movies.OwnStatus EQ "Own" </keep>
12
13 <segmentlist name="ratings" byequal="MyNum">
14 <rowlist name="dummy" />
15 </segmentlist>
16 </query>

17 </template>

```

When a `<segmentlist>` is declared, (line 13) it must contain either a `<rowlist>` or another `<segmentlist>`. However in this case, the HTML template only loops over the segmentlist (in order to create links to the five different star-rating pages). It is not looping over the rows in each segment, and in order to prove it, we named the `<rowlist>` `dummy`.

The `my-rating.extd` Template Declaration has a similar `<segmentlist>`, but here the `<rowlist>` has a meaningful name, because the loop over the `<segmentlist>` creates the five pages, and each page loops over the list of movies.

Amending the index Template file

As we mentioned earlier, the basis of the change to the Template file is to take out the loop through the movies themselves, leaving just the list of star-ratings.

4 - Complete/index.html

```

1 <H3>Here's what we have for ratings:</H3>
2 <table>

3 <tr valign="top" LOOP=ratings>
4 <td width="200">
5 <P><a href="[[URL(ratings.MyNum)]]-
  star.html">[[ratings.MyRating]]</a>
6 </td>
7 </tr>

8 <tr>
9 <td>
10 <P><a href="movie-list.html">Show me everything!</a>
11 </td>
12 </tr>
13 </table>

```

This time there will only be one table, with a loop listing all the genres first, and at the bottom a (hand-coded) row for the final “type” which is a list of all the movies.

So here’s the segment loop starting.

Closing the row—and therefore the loop which started in the begin row tag.

Now the final hand-coded row, with a link to the detail page that will contain a list of all the movies in the database.



If we were to render and display the new index page at this point it would look something like this:

Here's what we have for categories:



Okay, now we need to work on the detail page.

Creating the star-rating page Template Declaration file

In the previous segmented list example, we had a double loop with an outer loop around a heading and table, and an inner loop within the table, generating the list of movies. Now we want each page to have just one heading, and one table, containing just the movies in this category. The segmentation is still going to need the double loop of course, but this time we are moving the outer loop from the heading and table and instead we'll loop around the file name in the Template Declaration, so that rather than producing multiple headers we'll produce multiple files. In other words, the star-rating files are created from a segmented list where instead of starting a new table, each segment starts in a new file.

	3 - Segmented/index.extd	4 - Complete/my-rating.extd
1	<code><?xml version="1.0"?></code>	<code><?xml version="1.0"?></code>
2	<code><!DOCTYPE temp... "jar:/Template.dtd"></code>	<code><!DOCTYPE template SYSTEM "jar:/ Template.dtd"></code>
3	<code><template></code>	<code><template></code>
4		
5	<code><!-- One page of Movie Titles --></code>	<code><!-- One page per Star Rating --></code>

6	<code><output file="index.html" /></code>	<code><output file="[ratings.MyNum]-star.html"/></code>	The output statements change because now we want to create a series of output files, using a substitution, as we did before when we added the movie details page at line 5 on page 37. Notice in our new file that <code>ratings</code> is referencing the <code>segmentlist</code> at line 11.
7	<code><!-- List of Mov... --></code>	<code><!-- List of Movies seg'ed by Star Rating--></code>	
8			
9	<code><query table="movies" sortby="MyNum descending, Title"></code>	<code><query table="movies" sortby="MyNum, Title"></code>	In the index file the contents are segmented (line 11) by <code>MyNum (desc)</code> and within it, sorted by <code>Title</code> . We know that the <code>MyNum</code> loop is handled in the Template file.
10	<code><keep> movies.OwnStatus EQ "Own" </keep></code>	<code><keep> movies.OwnStatus EQ "Own" </keep></code>	In our new file, the query is the same...
11	<code><segmentlist name="ratings" byequal="MyNum"></code>	<code><segmentlist name="ratings" byequal="MyNum"></code>	... but this time the <code>segmentlist</code> is referenced by the <code>output</code> statement in line 5 which makes the outer loop.
12	<code><rowlist name="movierow"/></code>	<code><rowlist name="movierow" /></code>	Same rowlist
13	<code></segmentlist></code>	<code></segmentlist></code>	All's well that ends.
14	<code></query></code>	<code></query></code>	
15	<code></template></code>	<code></template></code>	

Creating the detail Template file

This page will contain many of the elements from the original detail page, with the important difference being that we need to reduce the list to just the movies in the selected category. This file will be used repeatedly by the Template Declaration above, where the `output` statement will call on the file each time it needs a new segment. So here, where the segments we want are the “star”



categories, we are expecting five segments—the five “star” categories—and therefore this file will be used five times.

4 - Complete/my-rating.html

```

1 <html>
2 <head>
3 <meta http-equiv="content-type" content="text/html; charset=ISO-
  8859-1">
4 <title>[[ratings.MyRating]] Movies</title>
<link rel="stylesheet" href="default.css">
  <base target="_top">
</head>

<body bgcolor="#ffffff">
5 <h1>[[ratings.MyNum]]-star ([[ratings.MyRating]]) Movies</h1>

6 <p><a name="[[ratings.id]]"></a>
7 <table LOOP=movierow >

8 <tr valign="top">
9 <td>
10 <p><a href="detail-
11 [[URL(movierow.id)]] .html">[[movierow.Title]]</a>
12 </td>
13 <td><p class="listing">[[movierow.Copyright]]</td>
14 <td><p class="listing">[[movierow.MPAA]]</td>
15 </tr>
16 </table>

17 <p><a href="index.html">Back to Home</a></p>

18 </body>
19 </html>

```

The browser’s title bar will display the genre type.

The main page header also shows the segment genre (the number of stars).

The inner loop begins (recall that the outer loop required for segmenting is now in `my-rating.html.extd` above, where it forms part of the output statement).

All’s well that ends.



We're done. Render the files and check it out!



Conditional Commands

There are several situations that come up frequently when generating content “automatically” and for which Exclamation has answers:

- A field has a label, such as Director: and a substitution such as `[[movierow.Director]]` but this particular instance of the field is empty. It would be nice to eliminate the Director: text when this occurs. The `VANISH` command provides this capability.
- You are using a loop command to list a series of items, and you want to place a separator (a comma “,” for example) between each item. This is done using the `BETWEEN` attribute. The `BEFORELAST` attribute can be used to provide a different separator between the next-to-last and last items (for example “, and”).
- You want to display the substitution text only if some other criteria is true (or false). This is done using the `IF` or `IFNOT` command attribute.

The sample files used in this section are based on those in the “Site Design/Advanced” folder.

Vanish

The Vanish tag allows content to be deleted if related substitution is missing. For example, the movie database includes ratings by the Motion Picture Association of America (MPAA). Sometimes this rating comes with a reason, for example “PG: for some scary moments some creature violence and mild language” but often it does not. The movie database stores this information in two fields, `MPAA` and `MPAAReason`. On the movie detail page, these two fields are separated by a colon (“:”). The colon therefore appears whether or not there is any content in the `MPAAReason` column. Vanish will allow us to eliminate the colon when we do not need it.



Element

Vanish can actually make content disappear in three different ways. The first option is ELEMENT (which is the default). Here's an example:

```
<td VANISH=ELEMENT> : [[movierow.MPAARReason]]</td>
```

This means that if the substitution is empty, Exclamation will eliminate not just the content, but the tag itself is deleted. This is not what we want here, because we need the empty <TD></TD> tag to ensure the integrity of the table. On the other hand we could equally well use this code:

```
<td>[[movierow.Rating]]
  <tbl VANISH=ELEMENT> : [[movierow.MPAARReason]]</tbl>
</td>
```

This achieves the desired result by eliminating the <tbl> tag and its contents if the substitution is empty, but not touching the empty <TD></TD> tags.

Content

Here's another example:

```
<td VANISH=CONTENT> : [[movierow.MPAARReason]]</td>
```

This means that if the substitution is empty, Exclamation will eliminate everything enclosed by the tag, but leave the tag itself. The detail-[[movierow.id]].html sample file shows this code at work. Some movies



have been given a personal star rating, and some have not.

But if there's no rating, there's also no heading.



Tag

The last option is `TAG`. This is the inverse of `CONTENT`: it eliminates the tag but leaves the content. This time the substitution is part of the tag. The most common use for it is in the anchor tag, so that if the `HREF` does not resolve, the link is eliminated, but the content which was to be linked remains. For example:

```
<h1>
  <a vanish=tag href="[ [URL(movierow.Web) ] ].html">[ [movierow.Title] ]</a>
</h1>
```

This code adds a hyperlink to the Movie Title header, if (and only if) the `movierow.Web` column is not empty for this movie. (Note that this example is not used in the sample files.)

Subtables

To look at subtables, we'll return to the cast entry in the Content Declaration file. `Castmember` as usual, is a column in the database table, but this column is treated differently in the Content Declaration file:

```
<column name="id" type="integer"/>
<column name="Title" type="safetext"/>
...
<subtable name="CastList" separator=",">
  <column name="CastMember" type="safetext"/>
</subtable>
<column name="Category" type="safetext"/>
<column name="SubCategory" type="safetext"/>
```

Note that the Movie project's Content Declaration file contains two entries for the cast information:

```
<column name="Cast" type="safetext"/>
<subtable name="CastList"
  separator=",">
  <column name="CastMember"
    type="safetext"/>
</subtable>
```

This is not a real-life situation. It has been done so that the same Content Declaration can be used throughout the book. We used the simple `Cast` column in previous examples, now we're going to use the more powerful `CastList`. Normally, only one would be needed, and most developers opt for the subtable because of its flexibility.

...

The `Cast` column contains a comma-separated list of actor names. Although the separator happens to be a comma, it could be any character: the `separator` attribute tells the subtable which character or string of characters to use. These characters are stripped from the string, so the result will be a long string of names, with no separator. The next topic, “Between and beforelast” explains how to exploit this feature, by replacing the missing separator with characters or code of your own.

5 - Advanced/Index.extd	detail-[[movierow.id]].extd	
<p>1 <code><query table="movies" sortby="MyNum descending, Title"></code></p> <p>2 <code><keep> movies.OwnStatus EQ "Own" </keep></code></p> <p>3 <code><rowlist name="dummy" /></code></p> <p>4 <code></query></code></p> <p>5</p> <p>6</p> <p>7</p>	<p><code><query table="movies" sortby="Title, Category"></code></p> <p><code><keep> movies.OwnStatus EQ "Own" </keep></code></p> <p><code><rowlist name="movierow" /></code></p> <p><code></query></code></p> <p><code><query list="movierow" subtable="CastList" ></code></p> <p><code><rowlist name="CastRow" /></code></p> <p><code></query></code></p>	<p>The query. The only difference being that we don't need to sort the output this time.</p> <p>End query statement.</p> <p>The second query is for the subtable. Notice the different syntax for the query: it names the rowlist from the previous query as it's source, and then names the subtable within the rowlist that it wants to use. (Remember that the sublist was named in the Content Declaration file.)</p> <p>Like “normal” queries, the subtable query has its own rowlist.</p> <p>End second query.</p>

Here's the call in the HTML template:

```

5 - Advanced/detail-[[movierow.id]].html
1 <td valign="top">
2 <p>

```

```

3 <tlb loop="CastRow">[[CastRow.CastMember]]
  <between>
  <br>
  </tlb>
4 </p>
5 </td>

```

<between> will place a
 tag between each of the items in the list.

Between and beforelast

To illustrate BETWEEN and BEFORELAST we're going to return to the subtable which lists out the cast members. The BETWEEN statement adds a line break between each of the actors, thereby turning the list into a column (in the detail-[movieid].html file):

```

<td>Cast: </td>
  <td><tlb loop=CastRow>[[CastRow.Cast]]<between><br></tlb></td>

```

BEFORELAST is generally used in comma-separated lists. As another example (not in the sample file) we could list all the categories:

```
<p>Back to this genre page:
```


```

<tlb LOOP=genresegment between="; " beforelast="; or "><a href="detail-
[[URL(genresegment.Genre)]] .html">[[genresegment.Genre]]</a><tlbend>.

```

```
Or return to the <a href="index-complete.html">complete movie list</a>.
```

Remember that spaces (or lack of them) between the HTML tags are critical. If the <tlbend> were not touching one another (for example if the <tlbend> were on the next line) HTML would add a space after each category name, which would separate it from the semicolon in the between attribute. Once the spacing is correct, the list is generated the way we want to see it:



```

Back to this genre page: Action; Animation; Comedy; Drama; Foreign; and Thriller.
Or return to the complete movie list.

```

To if or to ifnot, that is the question

Any tag can contain an `if` function. If the clause is true, then the contents of the tag are read, if not, the contents are ignored.

Here we'll use the `if` function to populate the sub-category column of the category pages. Recall that the page lists, say Action movies, and that movies can get into the action category either because their `Category` is `Action`, or because their `Subcategory` is `Action`. For example:

Movies	Category	Subcategory
Die Hard	Action	Comedy
Silverado	Western	Action

Therefore we don't necessarily want the page's `Subcategory` column to contain the content of the movie's `Subcategory` field, we want the content to be whichever field does not say `Action`:

Action Movies	SubCategory
Die Hard	Comedy
Silverado	Western

The following pair of `if` statements are mutually exclusive. If the page category matches the movie's subcategory, the top statement disappears. If it matches the movie's category, then the bottom one disappears. Whichever is left generates the required entry for the page's subcategory field:

```
<tlb if="Category.Name EQ movierow.Category">[[movierow.SubCategory]]</tlb>
<tlb if="Category.Name EQ movierow.SubCategory">[[movierow.Category]]</tlb>
```

Advanced topics

This section gathers a selection of unconnected challenges and their Exclamation solutions.

Sorts and other live action on pages

Exclamation prides itself on its static pages, but what if you want the user to be able to, say, sort the columns on a page? No problem. Typically there are a limited number of columns that need to be sorted, so generate all the different versions as static pages.

Note that if you are familiar with Cold Fusion, ASP, or any other sort of coding that can perform this task by accessing the database tables directly, there is nothing stopping you from doing that. Of course, this means maintaining the database on the public server, and all the complications that that implies.

List order in segmented lists

Most of the time the alphanumeric sorting in segmented lists is what you want, and most of the rest of the time, using the descend qualifier fixes any problems. But occasionally you need to specify a specific order, and to do that, we need to return to the database.

Suppose, for example, that we wanted to segment a list by MPAA rating. An “age” ranking makes much more sense here than an alphabetical ranking: G, PG, PG-13, R, NC-17, not G, NC-17, PG, PG-13, R. The solution is simple, and once set up, requires no extra effort or knowledge on the part of the content contributor.

Add a field the database called, for example `MPAARank`. This field takes its content from the `MPPA` field. So if the `MPAA` field’s content is `G`, then the `MPAARank` field is automatically filled with `1`. Using FileMaker Pro as an example, this is done by making the field a calculation, where the calculation might look like this:

```
If (MPAA="G", 1, If (MPAA="PG", 2, If (MPAA="PG-13", 3, If (MPAA="R", 4, If (MPAA="NC-17", 4, 0))))))
```

Wherever you need to rank the MPAA ratings, use the `MPAARank` column instead (remembering to use the original `MPAA` column whenever the content needs to be displayed on the screen).

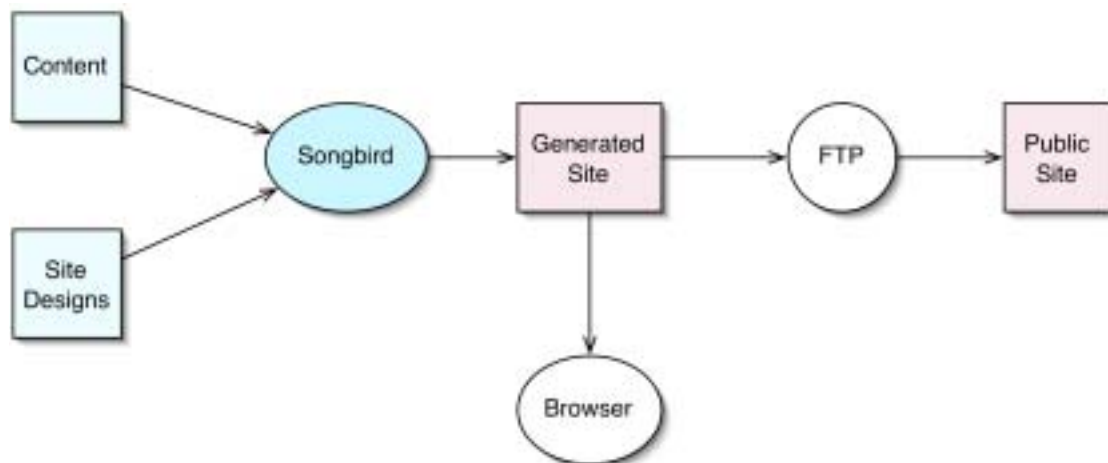
CONDITIONAL COMMANDS

Advanced topics

-
-
-
-

Publishing: Uploading and Viewing

If you've been following the examples through the book, you already have some experience with the steps up to "Generated Site" in the diagram below. If you just want to finish up, skip to "Previewing the results" on page 62.



This chapter describes the publication process, from start to finish, which also gives us an opportunity to look at the Exclamation user interface in greater detail than previous chapters have allowed.

The diagram above shows the basic steps:

- Editing the content
- Editing site designs
- Generating the site for previewing
- Using FTP to upload the site to the hosting service



Editing content

The content is maintained in one or more database tables which become the tab-delimited files used by Exclamation. For convenience, you can open the content-editing application by clicking on the Edit Content button. (You can also open the application from the Content list, by double-clicking on the database file.)



Editing the design

The Template Declaration and Template files are also maintained using third-party tools. As long as your editor has claimed ownership of your files, double-clicking on a file in the Site Design list will open the application as you would expect.



Some editors, for example Adobe GoLive, use project/site files. If you click on the Edit Design button, these editors will open the current site. Editors such as Macromedia Dreamweaver, which do not use these files will, as usual, open the last selected site.

Generating the pages

Recall from “Site Design” on page 22 that the Exclamation Site Design and Content folders contain all the files needed to create the Web site, but they now need to be processed:

- Content Declaration and the tab delimited content file(s) in the Content folder
- Template Declaration and HTML layout files, plus any hand-crafted files and any other files such as images in the Site Design folder.

Up until now, all we’ve done to generate the pages is to click on Make All Pages, but there are a couple of shortcuts that are extremely useful as the site gets larger, or when you are debugging.

First, you can generate only the files resulting from one or more specific Template Declarations. To do this, select the Template Declaration (or multiple

declarations) from the Site Design list then choose Project > Make Pages From Template. Alternatively, you can also select sample files from the Preview list. In either case, Exclamation will only generate the pages related to the relevant Template Declarations. This is most useful when you have other templates that generate large numbers of output pages.

Double-clicking on the page will bring that page up in the default third-party tool. Notice here that Exclamation is smart enough to bring up HTML pages in your editor if you have the Site Design list open, but in your favorite browser if it is selected from the Preview list!

If you've been making pages piecemeal... Whenever you think you are ready to upload the pages to the production Web site, it is a good idea to clear the pages in the Preview folder (Preview > Clear Preview Folder...) and Make All Pages once more. This way you will be sure that:

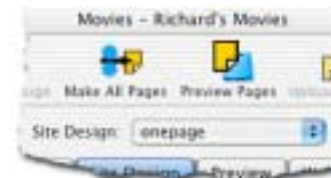
- make sure that content is up-to-date on all pages
- these are all the latest versions of the pages
- the current Site Design folder contains all the files needed to build the entire Web site. It adds all the hand-crafted files, so that the Preview folder contains all the files for the Web site

If there are problems with making the Web site, a log window opens to list them. The error messages endeavor to pinpoint the problem and its location.

There is a Troubleshooting section in the Reference Manual that explains some of the more complicated issues, but this section should help you to figure out most of them.

Maintaining multiple designs

Note that this is not the same thing as maintaining multiple projects. Multiple designs all use the same Content database. They are variations on the same basic Web site. For example you might have a summer and a winter look for the Web site. Multiple designs also make it easy to maintain an existing look while you work on a new one. When the new one is finally ready it is a trivial matter to publish the new design and replace the old one.



The movie project is a multi-design project, so if you've been following along through the rest of this book, you are already familiar with switching from one design to another using the Site Design pull-down menu.

A design is automatically added to this menu whenever you add a new sub-folder to the Site Design folder. Exclamation's Site Design and Preview lists always reflect the contents of the currently selected Design.

Previewing the results

At this point the Web site is ready to be previewed. Click on the Preview button, and the default browser will open, displaying the `index.html` page from your Preview folder. If your home page is not called `index.html` you can set the correct name in the Configuration dialog (which we looked at in "Using Exclamation to create the project" on page 31).

At first a quick tour through the pages will reveal any problems. Like any other programming or Web development project, things have a way of not looking quite the way you expected, and iterations are necessary. Indeed, like those other projects, making small incremental steps and testing each change as you go is a strongly recommended approach.

Since Exclamation reliably inserts page-to-page links, checking a few representative pages is sufficient to verify that the site is complete and correct.

Uploading to the hosting service

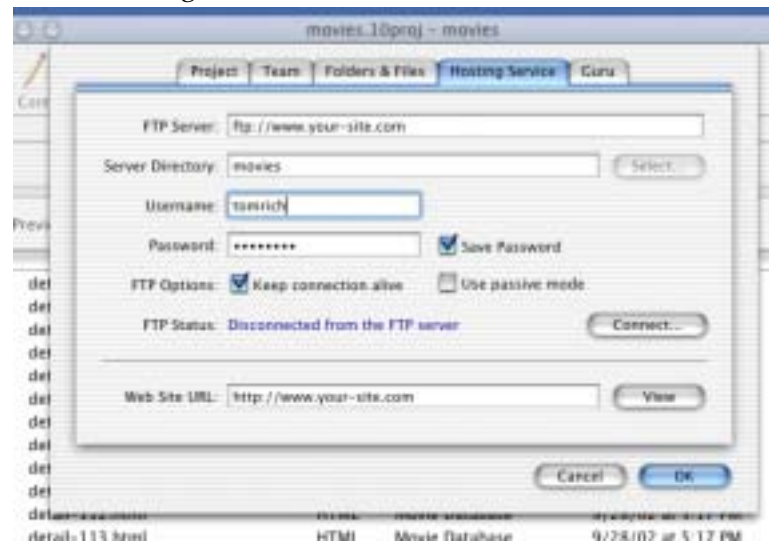
It is good practice (and in many cases essential) to maintain at least two versions of the Web site—the one you are working on and the publicly accessible one. Exclamation assumes that you have both. The public, or published site needs to be available to the intended audience, typically on the public Internet, and is often hosted by a remote hosting service that guarantees the server will stay up. To this end, Exclamation understands the concept of previewing on a local computer, and uploading (or publishing) to a remote one. Exclamation can be taught how to access the remote site using, say, FTP.

This major upload only happens the first time the site is uploaded, when every file must be uploaded. After that, Exclamation only uploads the files that have changed since the previous upload.

There are two procedures we need to discuss here: one-time provisioning of host server information, and the uploading process itself.

Configuring Exclamation with hosting service information

The hosting service is configured from the Project > Configure... > Hosting Service dialog:



Your service provider should have supplied the FTP server address (and user name) you need to use. Although the address is frequently in the format used in the example above, it is by no means always the case.

FTP's passive mode is off by default. If you are behind a corporate firewall you will probably need to turn it on. If you experience FTP problems (for example if uploads seem to "hang") these can often be fixed by using passive mode.

Uploading

Finally after you are satisfied that the site passes all your quality controls, it is time to upload it to your hosting service. Once the connection is established, uploading is as simple as clicking on the Upload to Web button. Exclamation will attempt to upload all the files, which if there are a significant number of files can take a significant amount of time. Exclamation keeps track of the uploading, so if you cancel in the middle of the process, or if the session is disconnected, the next upload will pick up from where it left off when the connection is re-established. Remember that this massive upload only happens the first time—after that Exclamation only uploads the updated files.

Viewing the completed, hosted, Web site

Once Exclamation has completed the upload, click on the View Web button. Exclamation will open your default browser, at the URL you provided in the Web Site URL field.

In theory, if you tested the site thoroughly while it was in the Preview folder (your staging site) then it should work just as well on the production server. Nevertheless it is always a good idea to retest on the production server.



Index

A

Adobe GoLive 8, 10
 templates, compared to Exclamation 20
ASP 10

B

Bare Bones Software BBEdit 10
beforelast 51, 55
between 51, 55

C

CMS
 see also content management system 7
Cold Fusion
 See Macromedia Cold Fusion
Content 22
content
 attribute of vanish command 52
 filtering 27
Content Contributors 9
Content Declaration 14, 19, 60
 creating 25

 field, in Options dialog 31
content files 19
Content folder 60
 description of contents 22
content management system 7
creating
 Content Declaration file 25
 Exclamation project 31
 segmented list with multiple output files 44
 simple segmented list 39
 Template files 29
 Web site, overview of steps 23

D

database 11
databases
 dirty 27
Dreamweaver
 see Macromedia Dreamweaver

E

element
 attribute of vanish command 52

examples
 explanation of examples used in the book 15
Exclamation 31
 advantages 14
 compared to Macromedia Dreamweaver templates 21
 converting sites from existing files 15
 creating a project 31
 creating sites from scratch 15
 functionality overview 10
 introduction 7
 key features 19
 publishing 59
 site structure 21
 Web sites 19

F

files
 content 19
 regenerating 14
 tab-delimited 19
filtering content

See also Keep, Omit 27
 Folders and Files tab 31
 FTP 62

G

GoLive
 See Adobe GoLive

H

hand-crafted pages 14, 19, 20, 60
 hosting service
 server information 63
 uploading to 62
 HTML layout files 60

I

if, ifnot 51, 56

J

JSP 10

K

Keep 27

L

log page 61
 loop 30

M

Macromedia Cold Fusion 10
 Macromedia Dreamweaver 8, 10

templates, compared to Exclamation
 Macromedia HomeSite 10
 multiple output files 33, 44

O

Omit 27
 Options dialog page 31

P

pages
 hand-crafted 19
 Preview folder 14, 22
 description of contents 23
 field, in Options dialog 32
 previewing 62
 project
 Exclamation, creating 31
 file 20
 Project folder
 field, in Options dialog 31
 provisioning 63
 publishing 59

R

regenerating files 14
 Rendering 38, 50
 rowlist 12

S

segmenting 39
 list order 57

multiple output files 44
 simple 39
 Site Design 22, 60
 Site Design folder
 description of contents 22
 field, in Options dialog 32
 Site Managers 9
 site structure 21
 Subtables 53

T

tab-delimited 19
 tag
 attribute of vanish command 53
 Template 19, 35, 42, 46, 48
 compared to Macromedia Dreamweaver templates 21
 creating 29
 relationship with Template Declaration 13
 rendering 32
 Template Declaration 12, 19, 37, 41, 45, 47, 60
 relationship with Template 13
 tools 10

U

uploading 59, 62, 64

V

vanish 51
 content attribute 52





- element attribute 52
- tag attribute 53
- viewing 59
 - hosted Web site 64

W

- Web design tools 10
- Web Developer 8
- Web site
 - converting existing to Exclamation 15
 - creating using Exclamation 23
 - Exclamation 19
 - production 15
 - traditional creation vs Exclamation 7
 - viewing 64

•
•
•
•