AMERICA ONLINE
ADMINISTRATION GUIDE

PLWeb Turbo 3.04 Windows NT

February 1999

Administration Guide: PLWeb Turbo 3.04 for Windows NT
February 1999

Document Number: AOL-C-PLS-028
Build Number: T1337_128

Distribution: Authorized Licensees and Internal America Online Staff Only

Note: Anyone who has obtained this document from `http://www.pls.com/` is subject to the terms and conditions of the PLS Free Software License Agreement.

Published by
America Online, Inc.
22000 AOL Way
Dulles, Virginia 21066, USA
E-mail: support@pls.com

Printed in the United States of America

Table of Contents

List of Tables

Welcome

Welcome to the *PLWeb Turbo 3.03 Administration Guide*. This manual will guide you through creating a Web-based information retrieval system using PLWeb Turbo.

Organizational Overview

The *PLWeb Turbo 3.03 Administration Guide* is organized as follows:

Chapter 1. **"Understanding PLWeb Turbo"**
Provides guidelines for designing and rendering your overall information management and retrieval strategies.

Chapter 2. **"Migrating PLWeb Turbo 2.6 to 3.0"**
Describes the steps PLWeb Turbo 2.6 administrators need to take to update their templates to PLWeb Turbo 3.0 format.

Chapter 3. **"Express Database Setup"**
Presents a quick method for turning source text into PLWeb Turbo databases. This shortcut is meant to serve as both a quick introduction to the product—giving you something with which you can experiment immediately—and as a method to revisit once you have absorbed the basics of PLWeb Turbo technology and capabilities.

Chapter 4. **"Preparing Source Files"**
Guides you in source-file preparation, which includes gathering and naming your files, adding record and field markup, and formatting numeric data.

Chapter 5. **"Defining Database Attributes"**
Describes the process for creating and defining the database definition file.

Chapter 6. **"Setting Global Application Parameters"**
Provides information for editing the `plweb.conf` file, which contains the global list of databases, the global list of views, and other global configuration options.

Chapter 7. **"Creating and Configuring Views"**
Outlines the procedure for creating a view and configuring the `view.conf` and `viewusrs.tab` files.

Chapter 8. **"Creating and Updating a Database"**
Outlines procedures for creating, adding, deleting, and maintaining your PLWeb Turbo databases.

Chapter 9. **"Using Relative Links in HTML Files"**
Discusses the options PLWeb Turbo offers for resolving relative links in HTML files that are part of your database.

Chapter 10. **"Configuring the PLWeb Daemon"**
Provides information for configuring the behavior of the PLWeb Turbo daemon.

Chapter 11.	**"Customizing Templates"** Guides you through customizing the PLWeb Turbo interface.
Appendix A.	**"Configuring the Custom Library"** Provides information for customizing the PLWeb Turbo `libplcustom` library, and gives an example of a custom billing application.
Appendix B.	**"Understanding the Default Templates"** Lists the default templates and explains the HTML markup and PLWeb Turbo macros found within them.
Appendix C.	**"List of Macros"** Describes the format and function of all PLWeb Turbo macros.
Appendix D.	**"Query Operators"** Lists, describes, and provides examples of the PLWeb Turbo family of query operators.
Appendix E.	**"Customizing Retrieval"** Describes the customizable strings listed in `defaults.cpl` and discusses customization of the default stopword list (`stopword.cpl`).
Appendix F.	**"Command-Line Invocation"** Provides instructions for running PLWeb Turbo from the command line.

Assumptions

Tip

This guide is intended as a handbook for the site system administrator. Discussions assume knowledge of the Windows NT operating system, as well as fundamental Internet, HTML, World Wide Web (WWW), and Web browser concepts.

Typographic Conventions

You will find the following typographic conventions used throughout this guide.

Angle Brackets < >

Where angle brackets are used to denote variable command-line parameters, text within the brackets is meant to be replaced with a literal value. The brackets themselves should be deleted, unless otherwise noted.

Where angle brackets are used in standard HTML markup, the angle brackets denote literal HTML commands and should not be deleted.

Example

```
plremote -f <drive>:\<installDir>\plweb\myfiles <dbname>
```

would be

```
plremote -f c:\mydbs\plweb\myfiles traveldb
```

Square Brackets [ ]

Square brackets are used to denote optional command-line parameters. The brackets themselves should be deleted, unless otherwise noted.

Example

```
plremote [-v] [-f <filelist>] <dbname>
```

would be

```
plremote -v -f c:\mydbs\myfiles traveldb
```

Non-Proportional Font

A non-proportional font (e.g., `courier`) is used to denote directory, file, utility, and field names, macro formats, template markup, and command formats.

Example

```
plremote -f c:\mydbs\myfiles traveldb
```

Italics

Italics are used to denote special terminology, emphasis, and in some instances, cross references.

PLWeb Turbo-specific Terminology

You will find the following PLWeb Turbo-specific terminology used throughout this guide:

advisors Online “search assistants” that help users narrow in on the most relevant search results. The Relate Advisor suggests words that are related to the query based on co-occurrence within a database. The Fuzzy Advisor, also known as the Spelling Advisor, suggests words that are spelled similarly to the first word in the query. The Dictionary Advisor certifies the existence of a word within a database.

advisor screen template A default interface template that displays results when a user selects one of the Advisors in the PLWeb Turbo interface.

authentication The process of verifying the identity and privileges of users. In this context, HTTP daemon authentication facilities are used to assign users privileges at both the system and database-view level. A PLWeb Turbo administrator can set up daemon authentication to allow users access to some, all, or none of the content published on the system.

base reference A tag used in HTML documents to resolve relative links.

base URL A PLWeb Turbo-specific facility for resolving relative links. Whereas the HTML base reference resolves links on a document-by-document basis, base URLs provide global resolution by mapping file paths to URLs. Administrators can use this facility to map the relationship between path and URL once in a single file rather than place an HTML base tag in every document.

database directory The directory that contains files associated with a particular database, including the database index file. Often referred to as `<dbdir>`.

database index A file (`.pls`) that stores the number of occurrences and location of all words in the database source files.

database name The name of the database. Often referred to as `<dbname>`.

definition file An ASCII file (`.def`), accompanying every database, that contains some of a database's characteristics, including the names and attributes of fields, the existence of a thesaurus, etc.

document screen template A default interface template that displays a hit document after users click on a hit in the hitlist.

field A unit within a record or document to which one can restrict a search. For example, a database of news articles would benefit from records being broken down into fields such as date, byline, lead paragraph, and keywords.

field attributes Settings in the database definition file that determine how fields are indexed, searched, and displayed.

field markup Text delineators in a source file that mark the beginning of fields.

hitlist A list of documents that meet the search criteria specified by the end user in a query. The list may consist of document titles or document summaries, depending on how you configure the PLWeb Turbo interface. PLWeb Turbo has the capability of presenting each item on the hitlist as a link to the full contents of the document it represents. Also referred to as *search results.*

index Refer to the definition of *database index.*

installation directory The directory in which you install the PLWeb Turbo software. Many examples in this guide will provide paths relative to this directory. Most often referred to as `<installDir>`.

local document A document that resides on the machine that is running PLWeb Turbo.

macro Similar to a function, a macro is an element of PLWeb Turbo's templates that takes arguments, gets expanded with information provided by PLWeb Turbo, and returns an output string to the HTTP daemon. PLWeb Turbo fills in the information the macro represents when it reads the template. For example, when an administrator places a macro for the list of databases in the template, PLWeb Turbo will display the list of current databases in the interface.

query A word or phrase an end user enters to describe his/her area of interest.

query rule A string in the templates that determines how PLWeb Turbo will combine the contents of the query input field(s) into a single query. The query rule consists of a multi-variable expression that

gets expanded based on the query the user has entered in the query text box(es). The query rule can be configured to enable range and field-restricted searching.

record A document in a database. In a PLWeb Turbo database, an administrator can index files as individual records or he/she can add markup to a single text file to break it down into multiple records. A record is the equivalent of a single document returned in a hitlist.

record markup Text delineators in the source of ASCII or HTML files that mark the end of records. Adding record markup to a file effectively turns a single ASCII or HTML file into multiple records or documents.

remote document A document that does not reside on the machine running PLWeb Turbo.

search What users do to find information about a topic of interest. The act of submitting a query for results.

search screen template A default interface template where end users enter query terms that describe areas of interest. This screen can be configured by administrators to provide users with several searching options, such as range and field-restricted searching, the use of advisors, etc.

search results Refer to the definition of *hitlist*.

search results screen template A default interface template that controls the appearance and functionality of the hitlist—the list of results returned for a search.

source file A file that is indexed into a database. This file can be ASCII, HTML, or Adobe Acrobat. A source file may constitute a single record or several records in a database. Synonymous with *source document*. See also *record*.

stopwords Words that PLWeb Turbo ignores during the indexing process in order to prevent the retrieval of extraneous records. PLWeb Turbo is shipped with a default list of stopwords that can be configured to meet specific needs.

synchronize The act of updating the database to reflect changes to the file system. The `plsync` utility shipped with PLWeb Turbo synchronizes databases by deleting and adding files from the database to reflect the current state of specific directories on a system.

templates The HTML forms shipped with PLWeb Turbo that control the functionality and appearance of the PLWeb Turbo interface. Administrators can modify the default templates or create entirely new templates to fit specific needs.

view Multiple configurations and customizations within the same installation of PLWeb Turbo where you can define an interface, the set of users allowed access to that view (if HTTP daemon authentication is enabled), the set of databases available in that view, and various configuration parameters.

view selection screen template A default interface template that marks the starting point for all search sessions.

1. Understanding PLWeb Turbo

The following sections discuss the principal design elements you should consider before you begin to create your PLWeb Turbo search and retrieval system. These sections are meant not only to provide you with the philosophy behind PLWeb Turbo database administration but also to aid your thought process as you map out the design of your site.

1.1 PLWeb Turbo Architecture

1.1.1 The Daemon

PLWeb Turbo uses a client/server architecture. The HTTP daemon receives user requests and invokes the PLWeb Turbo client (`fastweb`). The client, in turn, sets up a communication channel with the PLWeb Turbo server (`plwebd`). The server is responsible for handling and responding to the user's request. It performs the requested action and then, sends the request back to the PLWeb Turbo client, to be returned to the users. The PLWeb Turbo server is invoked once, by the administrator and is then able to service user requests until it is shut down (again, by the administrator).

For further information on the PLWeb Turbo daemon, refer to Chapter 10.

1.1.2 The Views

Views are multiple customized configurations within the same installation of PLWeb Turbo. For each view, you can define an interface, the set of users allowed access to that view (if HTTP daemon authentication is enabled), the set of databases available in that view, and various configuration parameters. Because virtually all of the customization and most of the configuration is done at the view level rather than at the global level, users can experience what appear to be entirely different applications, depending on the view they are using. Views are lightweight in terms of disk usage, memory usage, and process overhead. As a consequence, you are free to create as many views as is desirable for the purposes of customizing your application. For further information on views, refer to Chapter 7.

1.1.3 The Templates

PLWeb Turbo's interface is controlled by a set of HTML-like forms called templates. These templates are the mechanism by which you build your application's functionality, and they are the mechanism by which you present this functionality to your end users. For further information on templates, refer to Chapter 11.

1.1.4 The Macros

The elements in PLWeb Turbo's templates that get expanded with information provided by PLWeb Turbo are called macros. Much like a function, a macro is an element of PLWeb Turbo's templates that takes arguments, gets expanded with information provided by PLWeb Turbo, and returns an output string to the HTTP daemon. PLWeb Turbo fills in the information the macro requests when it reads the template. For example, when an administrator places a macro for the list of databases in the tem-

plate, PLWeb Turbo will display the list of current databases in the interface. For further information on PLWeb Turbo macros, refer to Appendix C.

1.2 Selecting Your Administration Utilities

You can administer certain aspects of your PLWeb Turbo Web site with either command-line utilities (e.g., `plcreate`, `pldelete`, etc.) or Web-based tools such as EZ Admin and PLSpider. Because the command-line utilities are more comprehensive than EZ Admin, keep in mind that, if you choose to use EZ Admin, you may also need to use the command-line utilities for some of your administrative activities.

✉✉ **Note** ✉✉

Some administration activities are mutually exclusive; they are either command-line driven or Web-based. For example, adding binary files to a database can only be accomplished by using the `pladdsur.pl` command-line utility. Crawling and indexing documents that are accessible via the World Wide Web can only be accomplished using the EZ Admin Web-based application.

1.2.1 Command-line Administration

The traditional method of administering your PLWeb Turbo site is batch processing via command-line utilities. Although PLWeb Turbo comes bundled with a Web-based administration tool (as discussed under “Web-based Administration” on page 23), you will find that you may still need to use the command-line utilities to perform certain functions.

✉✉ **Note** ✉✉

When executing a PLWeb Turbo command-line utility, any relative paths passed will be considered relative to the directory from which you executed the utility.

The following table briefly describes each PLWeb Turbo command-line utility:

Table 1-1: PLWeb Turbo Command-line Utilities[a]

Utility Name	Description	For more information, refer to page
`pladd*`	Adds local files of any of the following file types to a database: ASCII, HTML, Adobe Acrobat, and other files types as listed under "List of Document Formats and Filters" on page 232. Do not use this utility to add remote files of any kind. Refer below to `plremote` for adding remote files.	94
`pladdsur.pl`	Adds unsupported local and remote binary files (e.g., `.wav`) to a database.	98
`plcreate*`	Creates a new database and adds it to the global list of databases.	91
`pldbinfo*`	Provides access to database and document information such as general database information, the list of files contained in the database, general document information, and the contents of a particular document.	100
`pldelete*`	Deletes documents from a database.	103
`plezdb.pl`	Automates creating the most simple database configuration. Provides a quick, streamlined process for setting up a database. Invokes the `plcreate` and `pladd` utilities.	43
`plmerge*`	Merges multiple databases into one database.	109
`plremote*`	Adds remote files of any of the following types to a database: ASCII, HTML, Adobe Acrobat and other files types as listed under "List of Document Formats and Filters" on page 232.	95
`plreorg*`	Removes unused space from a database's index file. Only required if you have modified the database extensively, and you have limited disk space.	110
`plrmdb*`	Deletes a database, including its index file and its entry in `plweb.conf`.	112
`plrmfile*`	Deletes source files from a database.	104
`plsync*`	Updates a database to reflect changes to the set of source files. Synchronizes databases with the specified file system by deleting and adding files from the database.	107
`plupdate*`	Updates records in a database.	105
`plverify*`	Verifies the integrity of a database.	100

a. Utilities that have an EZ Admin counterpart are denoted with an asterisk.

✉✉ Note ✉✉

You can use the command-line utilities and EZ Admin interchangeably for operations that are supported in both capacities.

Note

PLWeb Turbo administration utilities log their errors to `<drive>:\<install-Dir>\plweb\logs\admin_err.log`.

1.2.2 Web-based Administration

PLWeb Turbo is bundled with a Web-based administration application that is designed specifically for database administrators who are looking to reduce the cost and complexity of managing large public or private Web sites. This application—EZ Admin—allows you to simplify administrative tasks by allowing you to perform most functions through a Web interface from your Web browser.

In addition, EZ Admin offers Web-crawling capabilities that you cannot accomplish via the command-line tools. The PLSpider tool in PLWeb Turbo can crawl an Intranet for Web-accessible documents and include them in a PLWeb Turbo database. PLSpider is configurable via EZ Admin.

EZ Admin makes the following PLWeb Turbo functionality more convenient:

- Administering databases remotely.
- Creating and deleting views.
- Creating and deleting databases.
- Adding and deleting documents, both local and remote.
- Merging and reorganizing databases.
- Synchronizing databases.
- Inspecting databases and verifying their integrity.
- Viewing previously set parameters for databases.

To use EZ Admin, simply launch the application from the following URL:

`http://<node>:<port>/ez_admin-cgi/main.pl`.

Then, from the menu of options, select the operation you wish to execute.

Note

Instructions for using EZ Admin and PLSpider may be found in the online help for those applications. This guide does not discuss further the use of either EZ Admin or PLSpider.

1.3 Structuring Your Data

Consider first a typical user search session: a user submits a query of one or more words against a database or set of databases and retrieves a hitlist that includes HTML links to documents relevant to the query. If you choose, the list of documents returned for a search can be ranked in order of relevance to the user's query, with the most relevant documents at the top of the list. The user then follows the links to view one or more documents.

Now consider the question of structure: Should you create a single database or build a group of separate databases? Your database design process should start with an examination of your goals in creating a database. From there, it should evolve into a general plan for achieving those goals and accommodating the needs of the database users.

Central to this decision is consideration of how users are going to access the data. For example, if usage patterns indicate that users would routinely search separate databases simultaneously, you could facilitate more efficient retrieval by combining that total body of information in a single physical or virtual database.

A good way to determine the number of databases that you need is to study how the information is organized and used in a non-database environment: observe or interview users of the information in its current format. You might include the following questions in your research:

- How is the information currently stored and used?
- What is its original format and purpose? PLWeb Turbo accepts ASCII, Adobe Acrobat, and HTML.
- What improvements in searching capability is expected through use of PLWeb Turbo by the database users and administrators?
- Does certain data share commonly used terms for which you plan to add a thesaurus?

1.4 Adding Record Markup

When designing a database, you need to determine whether you want to divide your ASCII and HTML source files into multiple records or leave each source file as a single record. If your source files contain a lot of data, you will want to break them into multiple records. You can do this by adding record markup (an end tag) throughout your source files to denote the end of a record. If your source files contain relatively little data, you will probably find that a one to one, source-file-to-record correlation will work fine for your database. For further information on adding record markup to your source files, refer to “Adding Record Markup” on page 48.

1.5 Adding Field Markup

When designing a database, you need to determine if you want to add fields to your ASCII and HTML records. By subdividing records into multiple fields, you enable users to search on specific sections of the data, and you control display attributes. If your data is in a simple format and has no natural field breakdown, you may choose to have only one field in each record.

When considering field markup, using the following methods will help users get the most out of your database.

- Fields can mirror the structure of the source information. For example, a book-like database might have `TITLE`, `AUTHOR`, `CHAPTER`, `SECTION`, and `TEXT` fields.

- Fields can define record types. For example, consider that a company which sells games has a database that describes their products. Fields in the database can be used to create different record types that contain different, but possibly overlapping, sets of fields.

  For board and card games, the fields might include

  ```
  NAME
  AGE_GROUP
  GAME_TYPE
  NUMBER_OF_PLAYERS
  ```

  For computer games, the fields might be

  ```
  NAME
  AGE_GROUP
  GAME_TYPE
  COMPUTER_SYSTEM_SPECS)
  MEDIUM
  JOYSTICK_REQUIRED
  EDUCATIONAL
  ```

- For information on adding fields to your source documents, refer to “Adding Field Markup” on page 49.

1.6 Organizing Views

Views are a central concept of PLWeb Turbo administration, enabling administrators to group data into a subset appropriate for a given set of users and to administer multiple subsets on a single physical machine. Many features, such as interface layout, logging, user access, and the default search operator can be controlled at the view level (in the `view.conf` and `viewusrs.tab` files).

From a design perspective, you will want to determine which databases belong together based on common information sets and shared administration needs. These determinations will drive your PLWeb Turbo architecture in terms of the number of views you need, the set of databases appropriate for each view, and the extent to which you want to customize the interface for each view and the interface.

Suppose that you have information concerning holiday destinations and lodging, air travel schedules, and special promotions for travel agencies. Suppose further that you serve users who are travel agents and users who are travel clients. You might then envision two views: `agents-view` and `clients-view`. The `client-view` might allow a client to login and search on current promotional travel packages, while the `agents-view` might allow travel agents to login and search on future promotional travel packages that are not yet available to the public.

For specifics on creating views and managing their configuration files, refer to Chapter 7.

1.7 Adding Documents

You can add both local and remote documents to your PLWeb Turbo database. Whether they are local or remote, your files can be of the following formats: ASCII, Adobe Acrobat, and HTML. Of the previously listed formats, only ASCII and HTML documents may be subdivided into records and fields. For further information on record and field markup, refer to Chapter 4.

1.7.1 Local Documents

Local documents are stored and indexed locally. That is, they are stored and indexed on the machine that is running PLWeb Turbo. You can add local documents to a PLWeb Turbo database with the `pladd` utility as discussed under “Adding Local Files” on page 94.

1.7.2 Remote Documents

Because PLWeb Turbo’s administration tools can index documents that reside on a remote server (i.e., any machine that can be accessed by an HTTP client), you have the option of adding remote documents to your database using the `plremote` utility. Remote ASCII and HTML documents remain at their original site; Adobe Acrobat documents get copied locally. When documents satisfy a query, their titles appear in the hitlist just as titles for local documents do, and they are displayed by the browser seamlessly. For more information on administering remote documents, refer to “Adding Remote Files” on page 95.

1.8 Setting User Authentication

You may find that you wish to control users’ access your PLWeb Turbo site. User authentication is controlled both at the view level (in the `viewusrs.tab` file) and by the HTTP server in the form of a user login ID and a password. For more information on configuring user authentication, refer to “Configuring the viewusrs.tab File” on page 86 as well as your HTTP server’s administration documentation.

1.9 Configuring Logging

PLWeb Turbo supports logging at the view level. Logging is turned on in the `view.conf` file, which is discussed on page 83.

1.10 Configuring Billing

PLWeb Turbo supports both database-wide and document-specific pricing and billing. To set up pricing and database-wide billing, you can customize the `<drive>:\<installDir>\plweb\etc\price.conf` file (refer to page 162) and then use the default `CPRICE` custom billing macro (refer to page 193) to make the price of documents in the database visible to end users. To set up document-specific billing, you must include the `PRICE` field in your source files (refer to page 55) and enable the `PRICE` macro (refer to page 214).

1.11 Tuning Retrieval

You can alter certain characteristics of PLWeb Turbo (e.g., default search operator, thesaurus behavior, stemming, etc.) by changing settings in `defaults.cpl`, a text file located in the `<drive>:\<installDir>\plweb\cpl` directory that contains numbered strings of text, each of which dictates a database definition or software configuration default for the PLWeb Turbo installation site. While certain settings can be superseded at the database level, you can modify a few site-wide defaults by changing the corresponding lines in `defaults.cpl`. For further information on tuning retrieval, refer to Appendix E.

1.12 Creating a PLWeb Turbo Database: Checklist

The following list outlines the steps you need to take when building your PLWeb Turbo search and retrieval system. Use this list as a checklist.

1. Study your existing data and formulate a plan for structuring it—refer to page 24.
2. Select and name your source files—refer to page 48.
3. (optional) Add record and field markup—refer to pages 48 and 49.
4. Create the database definition file—refer to page 60.
5. (optional) Create one or more views and configure the view configuration files—refer to page 82.
6. Create the database using either the `plezdb.pl` or `plcreate` utility—refer to page 43 or 91.
7. Add files using either the `pladd`, `plremote`, `pladdsur.pl` or `plsync` utility—refer to Chapter 8.
8. Modify the mappings for relative links in HTML documents, if necessary—refer to page 118.
9. (optional) Set up user access—refer to page 86.
10. (optional) Set up logging—refer to page 83.
11. (optional) Set up billing—refer to pages 55, 162, and 214.
12. Configure retrieval—refer to page 236.

13.(optional) Configure templates—refer to Chapter 11.

2. Migrating PLWeb Turbo 2.6 to 3.0

If you were a user of PLWeb Turbo 2.6, you will find this chapter useful in not only understanding the differences between the previous version and PLWeb Turbo 3.0 but also in updating previous databases to function properly with PLWeb Turbo 3.0.

2.1 Product Improvements

Improved performance. PLWeb Turbo 3.0's *persistent server* reads all configuration files and opens all databases at start-up time, reducing processing and improving performance.

Increased customizability. The templates in PLWeb Turbo 3.0 are more powerful and flexible than those in version 2.6. The new templates allow you to create *as many templates as you need* and to use almost all macros on any template. In other words, you are no longer restrained to a finite set of templates with specific macros. In addition, PLWeb Turbo 3.0 enables you to build additional functionality into your templates with *custom macros.*

✉✉ **Note** ✉✉

To create custom macros, you must have knowledge of C/C++ programming.

Enhanced administration. The *Web-based administration* of PLWeb Turbo 3.0 makes administration easier and more intuitive. Anyone can be a PLWeb Turbo database administrator with this easy-to-use forms-based interface. PLWeb Turbo 3.0 also includes Web-crawling capability, so you can easily turn your vast Web site into a valuable, searchable resource.

2.2 New Features

The following features are introduced in this version of PLWeb Turbo.

System and Database Administration

- Persistent server.
- Web-based administration.
- Web-crawling capability.
- Improved database synchronization.
- Additional database administration features, such as database merging, integrity verification and more.

Templates and Macros

- Context-free templates and macros.

- Custom macro creation.
- Ability to add formatting to HTML documents through the templates.

Enhanced Support for HTML Frames

- Field sorting.
- Reverse-chronological sorting.
- Scripting capabilities.

Logging and Billing

- Customizable event logging.
- Sample billing application.
- Customizable billing facility.

2.3 Finding the Shared Libraries

The shared libraries, including the CPL shared libraries, have been moved to the `<drive>:\<installDir>\plweb\lib` directory.

2.4 Finding the Macro Handlers

The macro handlers are now built into a shared library separate from the PLWeb Turbo daemon executable, which is also stored in the `<drive>:\<installDir>\plweb\lib` directory.

2.5 Configuring View-Specific Settings

All settings that were configurable in the `<drive>:\<installDir>\plweb\etc\viewlist.conf` file under PLWeb Turbo 2.6 are now controlled by the `<drive>:\<installDir>\plweb\views\view.conf` file. For further information on the `view.conf` file, refer to page 83.

2.6 Moving Your List of Databases

PLWeb Turbo 3.0 no longer uses the database list file, `dblist.cpl`. Databases are now listed in the `<drive>:\<installDir>\plweb\etc\plweb.conf` file. You will need to list databases that you

created with PLWeb Turbo 2.6 under the `Databases` section of `plweb.conf` file, as discussed under “Databases Section” on page 77.

2.7 Displaying a Record’s Container Name in the Hitlist

Container name display is now determined by the presence of the `CONTAINER` macro within the `HIT-LIST` macro loop (as described on page 203). In previous versions of PLWeb Turbo, the display of container names was set as an optional definition statement in the database definition file (`.def`), as follows

```
hitlist=container
```

Although this method of displaying the container name is still supported, when upgrading to PLWeb Turbo 3.0, we recommend you remove this line from `.def` file and use the `CONTAINER` macro instead.

2.8 Adding Remote Documents

When adding remote documents to a PLWeb Turbo 3.0 database, you should use the new `plremote` utility (page 95) as opposed to `pladdsur.pl`. The `plremote` utility automates the following functions that are done manually with `pladdsur.pl`:

- Fetching of the URL.
- Generation of a local, surrogate file.
- Generation of a title for the URL.
- Generation of a summary for the URL.

2.9 Adding HTML Documents that Contain Relative Links

A new directory called `<drive>:\<installDir>\plweb\public-dbs` has been created specifically for storing HTML source files that contain relative links. The default `baseurl.map` file has a mapping to this directory, and the installation program adds a mapping to the directory in the HTTP daemon configuration file. This is to enable direct access to any HTML source files placed in the `public-dbs` directory or its subdirectories, thereby making the handling of links easier. For more information on adding HTML files that contain relative links, refer to Chapter 9.

2.10 Updating Templates

Prior to updating your templates, you may find it useful to skim the chapters and appendices in this guide that address templates and macros so that you have a basic understanding of the framework and functionality of the new templates.

Because PLWeb Turbo no longer predefines the set of templates or the order in which they are accessed, you must edit the templates to dictate the order in which they will be accessed. In order to ensure proper access, you must make the following changes to the default templates as shipped in PLWeb Turbo 2.6.

1. Make sure all templates are properly accessed—refer to the sections that follow.

2. Remove macros that are no longer supported and update templates to achieve the same result—refer to page 37.

3. Change the format of all macros that take arguments—refer to page 37.

2.10.1 Home Page (index.html)

Change the link from your home page to specifically point to the View Selection template.

1. Edit the following line in your home page:

   ```
   <A HREF="http:/plweb-cgi/fastweb.exe?viewform">
   ```

   to

   ```
   <A HREF="http:/plweb-cgi/fastweb.exe">
   ```

2.10.2 View Selection Screen Template (views.tmpl)

Update the headers in your `views.tmpl` template to use the new form header define a target of the operation.

1. Change the existing form header from

   ```
   <FORM METHOD=post ACTION="http:fastweb.exe?searchform">
   ```

   to

   ```
   <FORM METHOD=post ACTION="http:fastweb.exe">
   ```

2. Insert a hidden HTML tag to define the target template of the operation:

   ```
   <INPUT TYPE=hidden NAME=TemplateName VALUE="search.tmpl">
   ```

2.10.3 Search Screen Template (search.tmpl)

Update your `search.tmpl` template to define a target template of the operation.

1. Change the existing form header from

   ```
   <FORM METHOD=post ACTION="http:fastweb.exe?search">
   ```


to

```
<FORM METHOD=post ACTION="http:fastweb.exe">
```

2. Include a hidden HTML tag to define the target template of the operation within the form:

```
<INPUT TYPE=hidden NAME=TemplateName VALUE="prehit.tmpl">
```

✉✉ **Note** ✉✉

The target of this operation is now the `prehit.tmpl` template rather than the search results screen template or advisor screen templates directly. If you do not support advisors and your users always go from the search screen to the search results screen, you can replace the last line with

```
<INPUT TYPE=hidden NAME=TemplateName VALUE="hitlist.tmpl">
```

If you do support advisors, you *must* use the `prehit.tmpl` template. This template will determine what operation was performed and select the appropriate template to use.

2.10.4 Search Results Screen Template (hitlist.tmpl)

Update your `hitlist.tmpl` template to define new target templates.

1. Change the `{$SEARCHFORM}` macro to `{$TOSEARCHFORM}`, and include the target template:

```
{$TOSEARCHFORM: TARGET=search.tmpl}
```

2. Change the `{$PREVLIST}` macro to include the target template:

```
{$PREVLIST: TARGET=hitlist.tmpl}
```

3. Change the `{$NEXTLIST}` macro to include the target template:

```
{$NEXTLIST: TARGET=hitlist.tmpl}
```

4. Change the `{$LINK}` macro to include the target template:

```
{$LINK: TARGET=predoc.tmpl}
```

2.10.5 Advisor Screen Templates (dict.tmpl, fuzzy.tmpl, relate.tmpl)

Update the headers in your `dict.tmpl`, `fuzzy.tmpl`, and `relate.tmpl` templates to use the new form header and define target templates of their operations.

1. Change the existing form header from

```
<FORM METHOD=post ACTION="http:fastweb.exe?searchform">
```

to

```
<FORM METHOD=post ACTION="http:fastweb.exe">
```

2. Include a hidden HTML tag to define the target template of the operation

```
<INPUT TYPE=hidden NAME=TemplateName VALUE="search.tmpl">
```

2.11 Updating the Query Rule

The query rule no longer requires square brackets. For more information on creating the query rule, refer to page 141.

2.12 Updating Macros

In PLWeb Turbo 2.6, all arguments to macros were interpreted by position. To allow a growing number of arguments without forcing the administrator to supply all the arguments, PLWeb Turbo 3.0 interprets arguments by name only. Use the following conversion table to convert your macros from 2.6 to 3.0 format.

Table 2-1: Macro Conversion Table

PLWeb Turbo 2.6	PLWeb Turbo 3.0	Refer to
{$CHECKED}	{$IF}, {$ELSE}, and {$ENDIF}	page 205 page 199
{$DBNAME: <n>}	{$DBNAME: size=<output_size>}	page 195
{$DOC: <begin_tag>, <end_tag>, <field_name>}	{$DOC: BeginTag=<begin_tag>, EndTag=<end_tag>, field=<field>, highlight=<highlight_flag>, format=<format_flag>, default=<default_text>}	page 197
{$DOCID: <n>}	{$DOCID: size=<output_size>}	page 198
{$DOCSIZE: <n>}	{$DOCSIZE: size=<output_size>}	page 199
{$ITEMSFOUND}	{$HITSFOUND}	page 204
{$ITEMSRETURNED}	{$HITSRETURNED}	page 204
{$LINK}	{$DOCLINK: HRefOnly=<href_flag>, TARGET=<TargetTemplate>}	page 198
{$NEXTLIST}	$NEXTLIST: HRefOnly=<href_flag>, target=<TargetTemplate>}	page 211
{$NRCHECKED}	{$IF}, {$ELSE}, and {$ENDIF}	page 205 page 199
{$NRSELECTED}	{$IF}, {$ELSE}, and {$ENDIF}	page 205 page 199
{$NSCORE: <n>}	{$NSCORE: size=<output_size>}	page 212

Table 2-1: Macro Conversion Table

PLWeb Turbo 2.6	PLWeb Turbo 3.0	Refer to
{$PREVLIST}	{$PREVLIST: HRefOnly=<href_flag>, target=<TargetTemplate>}	page 214
{$QUERYBYEXAMPLE: <n>}	{$QUERYBYEXAMPLE: numterms=<num_terms>, HRefOnly=<href_flag>, target=<TargetTemplate>}	page 216
{$RANK: <n>}	{$RANK: size=<output_size>}	page 216
{$SCORE: <n>}	{$SCORE: size=<output_size>}	page 218
{$SCOREIMG: <x>, <y>, <left>, <right>}	{$SCOREIMG: height=<graph_height>, width=<graph_width>, LeftImg=<left_img>, RightImg=<right_img>}	page 219
{$SEARCHFORM}	{$TOSEARCHFORM: HRefOnly=<href_flag>, target=<TargetTemplate>}	page 226
{$SELECTED}	{$IF}, {$ELSE}, and {$ENDIF}	page 205 page 199
{$SUMMARY: <l>, <c>}	{$SUMMARY: sentences=<num_sentences>, maxchars=<max_chars>}	page 223
{$TERM}	{$DICTTERM}, {$FUZZYTERM}, {$RELATETERM}	page 196 page 201 page 217
{$TERMID: size=<output_size}	{$DICTTERMID: size=<output_size>} {$FUZZYTERMID: size=<output_size>} {$RELATETERMID: size=<output_size>}	page 197 page 202 page 217
{$TERMLIST}	{$DICTLIST}, {$FUZZYLIST}, and {$RELATELIST}	page 196 page 201 page 217
{$TITLE: <n>, <field_name>}	Refer to the {$TITLE} macro.	page 224
{$VIEW}	{$VIEWNAME}	page 228

2.13 Logging Errors

PLWeb Turbo logs errors to <drive>:\<installDir>\plweb\logs\error.log.

2.14 Searching from the Command Line

Unlike previous versions of PLWeb, only a single “operation” is supported in PLWeb 3.0. The operation specifier is no longer part of the command line (e.g., viewform, search, searchform, and getdoc are no longer valid operations).

Differentiation is achieved by using different macros in the targeted templates, rather than supplying a different operation code on the command line.

The arguments to the command-line invocation have also been unified. One set of arguments is supplied. Arguments are specified as a set of key-value pairs. All arguments can be omitted except the `TemplateName` key and value, unless their values are different from the default values. For further information on command-line invocation, refer to Appendix F.

3. Express Database Setup

Your PLWeb Turbo distribution includes an express setup utility, `plezdb.pl`, that automates database creation. This utility is located in `<drive>:\<installDir>\plweb\bin`.

3.1 Chapter Checklist

Use this checklist to ensure that you have performed the necessary steps when creating a database with `plezdb.pl`:

1. Check your environment variables—refer to page 42.
2. Run `plezdb.pl`—refer to page 43.
3. Enable searching from a Web browser—refer to page 43.

3.2 Checking the Environment Variables

Before running the `plezdb.pl` utility, perform the following steps to check that the PLWeb Turbo environment variables were set by the installation program during installation.

1. Click the START button.
2. From the SETTINGS menu, select CONTROL PANEL.
3. In the Control Panel window, double-click the SYSTEM icon.
4. In the System window, click the ENVIRONMENT tab.
5. In the System Variable scroll box, check the following environment variables:

   `PATH` should be set to `<existing_path>;<drive>:\<installDir>\<lib>`

   `PLWEB_ROOT` should be set to `<drive>:\<installDir>`

   `PLTEMP` should be set to `<drive>:\<installdir>\tmp`

   where

   - `<existing_path>` is the path that existed prior to you installing PLWeb Turbo.
   - `<drive>` is the drive on which you installed PLWeb Turbo.
   - `<installDir>` is the directory in which you installed PLWeb Turbo.

3.3 Running plezdb.pl

Once you have your environment configured properly, you are ready to execute `plezdb.pl` from the `<drive>:\<installDir>\plweb\bin` directory as follows.

Format

```
perl plezdb.pl[-v] [-vr] <fullname> <src_path> [<file>]
```

Table 3-1: Command-line Parameters: plezdb.pl

Flag/Argument	Description	Explanation
`-v`	Verbose	Causes verbose output of all the commands used to create the database and index your source file.
`-r`	Recursive	Causes `plezdb.pl` to look recursively down from the `<src_path>` and add all of the files in the directory tree rooted at `<src_path>`.
`<fullname>`	Path to the Database	Specifies the location of the new database directory. The `<dbdir>` will contain your `.def` and `.pls` files. If you do not create a `.def` file in this location for `plezdb.pl` to use, a default `.def` file will be created for you. For information on creating the `.def` file, refer to "Creating the Database Definition File" on page 60.
`<src_path>`	Path to Source File	Specifies the location of the source file you wish to index. The file in the immediate directory will be indexed; use the `-r` option to index the entire subtree.
`<pattern>`	File Name Patterns	Specifies a pattern or series of patterns `plezdb.pl` should use to determine which files to index. File names matching any of the specified patterns will be indexed. If you do not specify a pattern, all files will be indexed. The patterns must be surrounded by double quotation marks. If you indicate more than one pattern, the patterns must be separated by a space. If you do not include the pattern parameter, any `.txt`, `.htm`, `.html`, or `.src` file found in `<src_path>` will be indexed.

Example

The following example would create a database called `film` in the `c:\myDatabases\filmdb\` directory by indexing all of the files listed in the files list. Files `film.def` and `film.pls` will now exist in `c:\myDatabases\filmdb`.

```
perl plezdb.pl c:\myDatabases\filmdb\film  c:\newsfeed\film_reviews film.lst
```

3.4 Enabling Searching from a Web Browser

After you have run `plezdb.pl`, to search your database from a Web browser, you must

1. Add the database to a view. For more information on creating and configuring views, refer to Chapter 7.

a. Create a subdirectory called <viewname> in the <drive>:\<installDir>\plweb\views\ directory, replacing <viewname> with the name of the view you are creating. You must have at least one view per PLWeb Turbo installation. (If you have only one view and use the default template for the view selection screen, the view selection screen template will be bypassed during searching.)

Note

The <viewname> directory does not have to be under the plweb directory structure. It can be in any location on any physical machine, as long as it is referenced correctly in plweb.conf. For more information on plweb.conf, refer to Chapter 6.

---

b. Create and configure a view.conf file in the <viewname> subdirectory using the guidelines under "Configuring the view.conf File" on page 83.

c. Add the <viewname> to the plweb.conf file using the guidelines under "Views Section" on page 78.

d. If you plan to enable access restrictions, create a viewusrs.tab file in the <viewname> subdirectory using the guidelines under "Configuring the viewusrs.tab File" on page 86.

e. If you intend to create view-specific templates for one or more views, create a templates subdirectory under the <viewname> directory, which is where you will store templates pertinent to that view. If you do not create view-specific templates, PLWeb Turbo will default to those listed in <drive>:\<installDir>\plweb\templates. For more information on configuring templates, refer to Chapter 11.

Tip

It is often easiest to configure the view.conf and viewusrs.tab files using existing ones, such as those in the sample-view, as models.

---

2. Restart the PLWeb daemon, as follows:

1. From the START button, select Control Panel.

2. In the Control Panel window, double click the SERVICES icon.

3. In the Services window, select the plwebd service and click STOP and then, click START.

After the server opens the local databases (you will get notification messages of this event), PLWeb Turbo will respond to user requests.

4. Preparing Source Files

4.1 Chapter Checklist

Use this checklist to ensure that you have performed the necessary steps for preparing source files:

1. Select your source files—refer to page 48.
2. Name your source files—refer to page 48.
3. If necessary, free up sufficient hard disk space for the temporary files that will be created during indexing—refer to page 93.
4. (optional) Add record markup—refer to page 48.
5. (optional) Add field markup—refer to page 49.
6. Format numeric data—refer to page 53.

4.2 Selecting Source Files

You can index into your database any or all of the following source file types:

- ASCII
- HTML
- Adobe Acrobat

4.3 Naming Your Source Files

When naming your source files, you may use whatever naming conventions make sense for your site; however, you must adhere to the following rules:

- Denote HTML documents in **one** of the following ways:
  - By the extension `.html` or `.htm`.
  - By the presence of an HTML tag (i.e., `<HTML>`) in the first 512 bytes of the source file.
- Denote Adobe Acrobat (PDF) files by a `.pdf` extension.

4.4 Adding Record Markup

Adding record markup is the process of placing text delineators in the source of ASCII or HTML files to mark the end of records. Adding record markup to a file effectively turns it into multiple records or documents.

You may add record markup to ASCII and HTML files as follows:

Table 4-1: Record Tag Format

File Format	End Tag Format
ASCII	`-end-`
HTML	`<!--plsfield:end-->`

4.4.1 Rules for Formatting Record Markup

When adding record markup to records, adhere to the following rules:

- Record tags must be flush left.
- Extra lines (new lines) after the last record marker will produce an empty record.
- (HTML only) Records may reference one another using HTML links and HTML `NAME` tags as long as the link is absolute. If the link is relative, it will not resolve. In addition, although absolute links will resolve properly, if the file (not the record) to which you are pointing is long in length, be aware that it may take awhile to display the file, because the entire file is being loaded by the browser.

4.5 Adding Field Markup

By dividing records into *fields*, you can enable your users to use field-restricted searching and to issue queries against the contents of a particular portion of each record—such as its title, author, date of publication, or any other field that you determine.

Field markup is supported for ASCII and HTML files only.

The contents of a source file without field markup will be indexed as a single record and occupy the first field in the `.def` file, which is `PLWeb/Default`. (For more information on the `.def` file, refer to page 60). The text used as a source file's hitlist summary will also be generated from this field in the document and will be displayed in the hitlist. For more information, refer to the `TITLE` macro on page 224.

4.5.1 Rules for Formatting Field Markup

When adding field markup to records, adhere to the following rules:

- Each record may contain up to 32,767 unique field tags; this is the maximum number of unique fields you can define for a database.

- Field tags must be formatted as follows:

Table 4-2: Field Tag Formats

File Format	Field Tag Format	Explanation
ASCII	-<field_name>-	<field_name> specifies the field name.
HTML	<!--plsfield:<field_name>-->	<field_name> specifies the field name.
HTML with Meta Tags	<meta name="<field_name>" content="<content>">	<field_name> is a field name, conforming to the guidelines described under "Adding Field Markup" on page 49, with the exception that, when defined in the .def file, <field_name> may only be given the S field attribute. For more information on field attributes, refer to Table 5-1, "Field Attributes," on page 61. <content> is the content of <field_name> that will be indexed into and retrievable from a database.

- Field tags must be flush left.
- A field tag must precede the information to be contained in the corresponding field (i.e., the field tag -DATE- must be placed before text that is to appear in the DATE field).
- A field tag does not have to be followed by a new line (the following examples use them for the sake of clarity).
- Source files may contain some or all of the fields declared in the definition file. For more information, refer to "Creating the Database Definition File" on page 60.
- Any content not explicitly assigned to a field (either the file contains no field tags, or content precedes the first field tag) will be indexed as if it were part of the first field specified in the database definition file, which is the PLWeb/Default field.
- Field tags can be in a different order from record to record; this is discouraged, however, because it may disorient users who become accustomed to looking in a given place for a given type of information.
- Field tags must contain field names corresponding to those defined in the database's definition file (.def). For more information on the .def file, refer to "Creating the Database Definition File" on page 60.
- Field tags can be repeated an unlimited number of times in a record, in any order. However, if these fields are displayable (i.e., if you have assigned them the D attribute in the .def file, which is discussed in Table 5-1, "Field Attributes," on page 61), highlighting inconsistencies may occur.
- A field name
  - May be up to 29 characters long, including punctuation.
  - May include the underscore (_) and the forward slash (/). It may not include other punctuation.

- May not contain spaces.
- Can be in upper-, lower- or mixed-case; however, case does not make a field name unique.
- Must begin with an alphabetic character; subsequent characters may be alphanumeric.
- May not begin with another complete field name (e.g., you could not define both `DATE` and `DATE_PUBLISHED` as field names for the same database).

- Field markup may not be placed in the middle of HTML title markup. Use the following format instead:

```
<!--plsfield:Title-->
<TITLE>Astronomy</TITLE>
<!--plsfield:Subject-->Astronomy
```

✉✉ Note ✉✉

If your documents do not include `<TITLE>` tag markup, PLWeb Turbo will add it to all documents, in order to have a displayable title in your Web browser. PLWeb Turbo will not add `<TITLE>` tag markup to HTML documents if the tag is already present in the source text. It is only necessary to add `<TITLE>` tag markup to your HTML source files if you want a title to appear in the Web browser title bar.

- When gathering text for title display, PLWeb Turbo will use the first of the following conditions that applies for the record:
  - (HTML only) Text defined within the HTML `<title>. . .</title>` tag is found within the first 512 bytes of the record. If an HTML title tag is found in the first 512 bytes of a document, no other text will be displayed in the hitlist, regardless of the presence of other fields with the `T` attribute in the `.def` file. For more information on field attributes, refer to Table 5-1 on page 61.
  - Text for a field defined with the `T` attribute, if it is found within the first 50 lines of the record. For more information, refer to Table 5-1 on page 61. For information on configuring the number of lines PLWeb Turbo will search to find a field defined with the `T` attribute, refer to “Setting the Maximum Number of Lines Scanned for the Hitlist Display Attribute” on page 238.
  - The first `<N>` characters or the first `<N>` lines of text in the file—provided they are non-blank. For more information, refer to the `TITLE` macro on page 224.
  - If none of these conditions apply for a record, the hitlist summary in PLWeb Turbo’s search results screen will, by default, return nothing at all. However, you can configure this behavior in the `TITLE` macro, as discussed on page 224.

Note

PLWeb Turbo does not report invalid field tags.

4.5.2 ASCII Example

```
-DATE-
19911219
-HEADLINE-
BCCI seen pleading guilty in agreement
-BYLINE-
Rob Wells
-DATELINE-
NEW YORK
-EDITOR-
Andy Livingston
-TEXT-
NEW YORK - Bank of Credit and Commerce International could plead guilty to...
-END-
```

4.5.3 HTML Example

```
<HTML>
<HEAD>
<title>BCCI seen pleading guilty in agreement</title>
</HEAD>
<BODY>
<!--plsfield:DATE-->
19911219
<!--plsfield:HEADLINE-->
BCCI seen pleading guilty in agreement
<!--plsfield:BYLINE-->
Rob Wells
<!--plsfield:DATELINE-->
NEW YORK
<!--plsfield:TEXT-->
NEW YORK - Bank of Credit and Commerce International could plead guilty to...
</BODY>
</HTML>
<!--plsfield:END-->
```

Note

For HTML source files, a `<!--plsfield:END-->` tag is only necessary for multi-record files.

4.5.4 HTML with Meta Tags Example

```
<HTML>
<HEAD>
<title>BCCI seen pleading guilty in agreement</title>
<meta name=”authors” content=”John Doe”>
<meta name=”keywords” content=”BCCI, bank, guilty”>
</HEAD>
<BODY>
<!--plsfield:DATE-->
19911219
<!--plsfield:HEADLINE-->
BCCI seen pleading guilty in agreement
<!--plsfield:BYLINE-->
Rob Wells
<!--plsfield:DATELINE-->
NEW YORK
<!--plsfield:TEXT-->
NEW YORK - Bank of Credit and Commerce International could plead guilty to...
</BODY>
</HTML>
<!--plsfield:END-->
```

✉✉ **Note** ✉✉

HTML meta tags are inherently non-displayable through browsers.

4.6 Formatting Numeric Data

Numbers in your source documents are added as-is to the database's index. This means, for example, if you had a document that contained the following date field,

```
<!--plsfield:DATE-->March 4 1998
```

the numbers `4`, `1998`, and the word `March` are added to the database index. Before being added, however, PLWeb Turbo pads the numbers with zeros in order that they be sorted properly (the zeros are otherwise ignored). So, the date `March 4 1998` is actually stored in the index as follows:

```
0000000004
0000001998
March
```

Like characters, numbers are recognized and stored in the database dictionary by their ASCII values. PLWeb Turbo determines the relative order of the ASCII values by the order in which they appear in the database's dictionary. Thus, a numeric term is evaluated as "less than" an alphabetic term.

4.6.1 Dates and Range Searching

Range searches can be performed on any term, be it a alphabetic, numeric, or alphanumeric. Range searches are most useful, however, against numeric terms (a.k.a., date range searches).

As an example, assume the above date March 4 1998 as well as content from several other documents has been stored in the database dictionary, as follows:

```
0000000001
0000000002
0000000003
0000000004
0000000012
0000000014
0000000030
0000001994
0000001995
0000001996
0000001997
0000001998
0019941212
0019950604
0019960930
0019970714
0019980304
100ft
31st
3rd
56k
5pm
8bit
90k
9am
9pm
9th
April
February
January
July
June
March
May
```

In addition, assume you wanted to search this database for all documents dated January 1998 through March 1998. If you looked at the DATE field in the documents, you would find quite a range in dates. For example, if you had five documents with the DATE field defined as follows

```
<!--plsfield:DATE-->March 4 1998
<!--plsfield:DATE-->December 12 1994
<!--plsfield:DATE-->June 4 1995
```

```
<!--plsfield:DATE-->September 30 1996
<!--plsfield:DATE-->July 14 1997
```

Query You Enter	Search Results	Explanation of Search Results
DATE=1998	All documents in the database that include 1998 in the DATE field, regardless of month.	Because you only specified the year for which to retrieve documents, all documents with that year in the DATE field (regardless of month) will be retrieved.
January<=DATE<=March	All documents from any year for the months that sort, in the dictionary, between January and March (July and June).	Because the year has been eliminated from the query, PLWeb Turbo searches for documents with dates January, July, June and March (refer to the dictionary listing above), regardless of year. Documents with February in the DATE field are not retrieved because February falls before January in the database dictionary.

As you can see, if you wanted to perform a range search for a document where you specify the year, month, and possibly even the day, having the source data in the format `March 4 1998` would not work well. Using the format YYYYMMDD would yield better results. If dates were stored in your documents as

```
<!--plsfield:DATE-->19980304
<!--plsfield:DATE-->19941212
<!--plsfield:DATE-->19950604
<!--plsfield:DATE-->19960930
<!--plsfield:DATE-->19970714
```

they would be added to the database as single terms, and they would be sorted properly. Issuing a date range search as follows

```
19980101<=DATE<=19980331
```

would retrieve all documents dated January 1 1998 through March 31 1998.

Caution

To use the `#date` function provided by PLWeb Turbo, your source documents **must** use the YYYYMMDD format. Your users, however, should not use this format when entering queries. For more information on the `#date` function, refer to Table 11-10, “Query Rule Functions,” on page 142.

4.6.2 Prices

In order for PLWeb Turbo to successfully execute searches on prices, you must store prices in their own PRICE field as an integer representing the smallest currency unit available (e.g., cents or pennies) for your locale. Thus, the price $1.50 would be stored in a record as 150.

For more information on building pricing into your application, refer to the PRICE macro on page 214, the CPRICE macro on page 193, and “Creating a Custom Macro” on page 162.

Example

```
-PRICE-
150
```


5. Defining Database Attributes

The database definition file (`.def`) is an ASCII file that describes some of the database's characteristics, such as fields, index size, and partitions. If you want to include rules for optional definition statements such as fieldlists, index size, thesaurus, or partitions (refer to “Adding Optional Definition Statements” on page 63), you must manually create the file *before* creating the database.

5.1 Chapter Checklist

1. Create the database definition file—refer to page 60.
2. Define fields and add field attributes—refer to page 60.
3. Add optional definition statements—refer to page 63.
4. Add comments—refer to page 73.

5.2 Creating the Database Definition File

The definition file should reside in the database directory (`<dbdir>`), which contains other configuration files associated with the database (such as its `.pls`, `.adf`, and `.acc` files). The `<dbdir>` can be either a previously existing or newly created directory. The definition file must end with the `.def` extension.

✉✉ **Note** ✉✉

Every PLWeb Turbo database *must* have an associated definition file, regardless of whether you are defining fields.

💣💣 **Caution** 💣💣

After you have created the database (refer to “Creating a Database” on page 91), if you delete, rename, or reorder the fields in the definition file you will have to recreate the database.

The definition file must take the following format:

Format

```
PLWeb/Default     (DST)
<Fieldname1>      (<FieldAttributes>)
<Fieldname2>      (<FieldAttributes>)
<Fieldname3>      (<FieldAttributes>)
<DefStatement>
# <Comment>
```

where

- PLWeb/Default *must* be the first field declared in the definition file with the attributes DST.
- <FieldnameN> is the name of a field contained in the database's records.
  - <FieldnameN> can be up to 29 characters long, including punctuation.
  - The underscore (_) and the forward slash (/) are the only punctuation marks allowed in <FieldnameN>.
  - <FieldnameN> may not contain spaces.
  - <FieldnameN> may be in upper-, lower- or mixed-case.
  - <FieldnameN> must begin with an alphabetic character; subsequent characters may be alpha-numeric.
  - <FieldnameN> may not begin with another complete field name (e.g., you could not define both DATE and DATE_PUBLISHED as field names for the same database).
  - <FieldnameN> must correspond to a field tag used in the database's source file(s).
  - Values for <FieldnameN> do not have to be in the order in which their corresponding fields appear in records.
- The attributes must be separated from the field name by a space or tab.
- A maximum of 32,767 fields can be defined for a database.
- All lines must be flush left.
- <FieldAttributes> is a string of abbreviations for field attributes, as explained in the following table:

Table 5-1: Field Attributes

Abbreviation	Name	Function
D	Display	The field will be displayed in PLWeb Turbo records.
T	Title	The field will appear in the record's hitlist title.
S	Search	The field will be searched by default, unless excluded by a field-restricted search. Fields without this attribute *will only* be searched when specified in a field-restricted search.
N	No Index	The field will not be indexed and therefore, will not be searchable.

  - At least one field with a D attribute must be defined.
  - A field that has the N attribute cannot also have the S attribute.
  - At least one field with the S attribute should be defined for each database; otherwise, all non-

field-restricted searches against the database will return no hits.

- At least one field with the T attribute must be defined for each database; if no such field appears within the first 50 lines of a record, the hitlist summary in PLWeb Turbo's search results screen will, by default, return nothing at all. However, you can configure this behavior in the TITLE macro, as discussed on page 224. The number of lines PLWeb Turbo will search to find a field defined with the T attribute is controlled by string 238 in defaults.cpl, as discussed under "Setting the Maximum Number of Lines Scanned for the Hitlist Display Attribute" on page 238.

- <DefStatement> is an optional definition statement that can define such database characteristics as thesauri or fieldlists. Optional definition statements may not precede field names and attributes. For further information on defining optional definition statements, refer to "Adding Optional Definition Statements" on page 63.

- <Comment> is an optional line of internal commentary for the file. Comments are recognized by a # character at the start of the line. Field names/attributes must precede comments; however, comments and optional definition statements may be in any order.

Example

```
PLWeb/Default     (DST)
Title             (DST)
Publisher         (DS)
Author            (DS)
Editor            (DS)
Text              (DS)
fieldlist=abstract title,author,publisher
# this database created by Herbert West on
# October 1, 1997
```

If you define fields for a database, we recommend that you list and describe them in the dbdef.html file. The dbdef.html file, which may be found in <drive>:\<install-Dir>\plweb\htdocs, contains information about each database.

5.2.1 Adding a New Field to an Existing Definition File

If, after creating a database (as discussed under "Creating a Database" on page 91), you find that you want to add a new field to your definition file (for instance, because you are adding new source files that contain the new field), you can do so by following this procedure:

1. Add the new field and attributes to the .def file *after* the last field listed, *before* any optional definition statements.

2. Add the new source files containing the new field to your database, using the instructions under "Adding Files" on page 94.

If, after creating a database, you find that you want to add a new field to your definition file, *and* you want to add the new field and its associated text to the "old" source files, you can do so by following this procedure:

1. Add the new field and attributes to the `.def` file *after* the last field listed, *before* any optional definition statements.

2. Follow the instructions described under "Updating Records" on page 105.

Example
As an example, assume your database contains information about a music store's inventory. Your `.def` file might originally appear as follows:

```
PLWeb/Default  (DST)
Title          (DST)
Composer       (DS)
Medium         (DS)
Product_Number (DS)
Price          (DS)
Genre          (DS)
fieldlist=Repertoire  Composer,Genre
thesaurus=PLWeb   c:\usr\thesaurus\classical.thes
```

You decide to add new documents to the database that contain a new field called PERFORMERS, so you add the new field to the `.def` file. Your new `.def` file appears as follows:

```
PLWeb/Default  (DST)
Title          (DST)
Composer       (DS)
Medium         (DS)
Product_Number (DS)
Price          (DS)
Performers     (DS)
Genre          (DS)
fieldlist=Repertoire  Composer,Genre
thesaurus=PLWeb   c:\usr\thesaurus\classical.thes
```

5.3 Adding Optional Definition Statements

By adding optional definition statements to a database's definition file, you can define the following characteristics specific to a particular database:

- **Fieldlists** Represent groups of fields that are regularly searched as a group. For more information, refer to "Defining a Fieldlist" on page 64.

- **Thesauri** Enable end users to use the *thesaurus* operator during searching. For more information, refer to "Creating a Thesaurus" on page 65.

- **Field Label Display** Overrides the display of field labels in the retrieved ASCII documents. For more information, refer to "Modifying the Display of Field Labels in Records" on page 69.

- **Partitions** Optimizes I/O. For more information, refer to “Optimizing I/O” on page 70.

- **Index Size** Controls how much information will be stored in the database index. For more information, refer to “Controlling Index Size” on page 72.

When adding optional definition statements to a database's definition file, follow these guidelines:

- Each optional definition statement must be flush left and begin on a new line.

- Optional definition statements can be in upper-, lower-, or mixed-case.

Sample Definition File with Optional Definition Statements

```
PLWeb/Default (DST)
Title      (DS)
Publisher  (DS)
Author     (DS)
Editor     (DS)
Text       (DS)
fieldlist=abstract title,author,publisher
partition=acctlaw.dic dic
partition=acctlaw.stk storage_keys
index_size=FULL
```

5.3.1 Defining a Fieldlist

Fieldlists represent groups of fields that are regularly searched as a group. A fieldlist is a single name (or alias) with which a user can refer to several fields. For field-restricted queries, a user can specify fieldlists in place of field names and thereby minimize typing in the PLWeb Turbo query box. The following example illustrates the format an end user would enter when searching with a fieldlist:

```
(Lovecraft Arkham Dunwich):abstract
```

If, for this example, `abstract` were a fieldlist defined in the current database for the frequently searched fields `author`, `publisher`, and `title`, this query would search only those three fields.

You can define one or more fieldlists for a database by adding the appropriate statement to its definition file, as follows.

Format

```
fieldlist=<listname> <fieldname>,<fieldname>...
```

The following table explains the parameters you may use when defining a fieldlist in the `.def` file:

Table 5-2: Optional Definition File Statement: Fieldlist

Argument	Description	Explanation
`<listname>`	Fieldlist Name	Specifies an alias for multiple fields. `<listname>` must conform to the same guidelines as field names and must be separated from `<fieldname>` by a space or tab.
`<fieldname>`	Name of a Field in the Fieldlist	Specifies the field that the fieldlist will represent when used in a query. A field name can appear in multiple field lists.

Example

```
fieldlist=abstract title,author,publisher
fieldlist=staff author,artist,editor
```

Note

Defining a fieldlist does not require that the associated database be reindexed; therefore, you can add, modify, or delete a fieldlist at any time after creating a database.

Note

When implementing fieldlists, we suggest you add an explanation of their use to the existing online help topics.

5.3.2 Creating a Thesaurus

You can create a thesaurus for PLWeb Turbo databases and thereby enable clients to use the thesaurus operator (see “Using a Thesaurus” on page 69), which replaces the query word to which it has been appended with synonyms from a predefined thesaurus file.

Note

Because concept searching and the relate advisor make it easy to identify and search with words that are related to query terms, you may find that you do not need to provide a thesaurus with your PLWeb Turbo databases. A thesaurus is ideal, however, for a database that includes many acronyms, because thesaurus searching can minimize the need to correctly interpret acronyms or manually enter the words that they represent.

Note

If you create a thesaurus for any PLWeb Turbo database, we suggest you add to the existing online help topics an explanation of the thesaurus operator, similar to that found at the end of this section.

Creating a thesaurus requires two steps:

1. Defining the thesaurus file—refer below.

2. Adding an optional definition statement to the `.def` file for the thesaurus—refer to "Defining the Thesaurus in the Definition File" on page 68.

5.3.2.1 Defining the Thesaurus File

The first step in creating a thesaurus is populating a text file with main entries and their corresponding synonyms.

Format

```
<main_entry>      <synonym>  <synonym>  <synonym>  . . .
```

The following table explains the parameters you may use when defining thesaurus entries in your thesaurus file:

Table 5-1: Thesaurus File Format

Argument	Description	Explanation
`<main_entry>`	Thesaurus Term	Specifies term for which synonyms are defined. This is also the term to which end users will append the thesaurus operator (@) when they do a thesaurus search. `<main_entry>` must be (1) flush left; (2) on a new line; (3) a single word; (4) in either upper- or lower-case (mixed case is not allowed); (5) separated from `<synonym>` by a space or tab; and (6) in alphabetical order, if you have more than one `<main_entry>` specified in your thesaurus file.
`<synonym>`	Synonym for Thesaurus Term	Specifies the replacement term for the main entry. During searching, when the thesaurus operator is appended to a search term that matches a `<main_entry>` in the thesaurus file, the search term is replaced with the list of synonyms for that main entry. `<synonym>` may be in mixed case. Multiple synonyms do not need to be in alphabetical order, but they must be separated by a space or tab. If your list of synonyms extends beyond one line, you must insert a tab at the beginning of the new line.

Note

Defining a thesaurus does not require that the associated database be recreated; therefore, you can create or modify a thesaurus at any time after creating a database.

Example

```
marx   (Groucho  Chico  Harpo  Zeppo  Gummo  Karl)
```

Based on the above example and assuming the default operator is OR, PLWeb Turbo would interpret the following query

```
Engels Marx@
```

as

```
Engels OR (Groucho OR Chico OR Harpo OR Zeppo OR Gummo OR Karl)
```

Tip

If you want a main entry to be included in a thesaurus search, you must define it as its own synonym:

```
vampire  (nosferatu  succubus  incubus  vampire)
```

Because using the thesaurus is like performing a search and replace, any search term that has the thesaurus operator appended to it will be replaced by its thesaurus file synonym list if it matches a main entry in the thesaurus file. You can use any search operator in the thesaurus file's list of synonyms.

Consider the following scenario:

1. Your thesaurus file is defined as follows:

```
bread          (wheat OR rye OR sweet OR sourdough@ OR (‘sticky buns’) OR (date
               adj nut) OR coffeecake OR bread)
cheese         {goat feta swiss blue havarti muenster cheese}
chocolate      (chocolate OR chip)
coffeecake     (coffeecake OR (coffee adj cake))
cookie         {sugar gingerbread chocolate@ oatmeal cookie}
sourdough      (sourdough OR (sour adj dough))
```

Note

If curly braces are used in the thesaurus file, PLWeb Turbo will use an OR operator between terms, regardless of the current default operator.

---

2. The user inputs the following searches (note that the letters *A*, *B*, and *C* are not part of the queries):

   A. `low fat cheese@`

   B. `cookie@ recipes`

C. `baking bread@`

Based on the above input and assuming the default operator is `OR`, PLWeb Turbo will evaluate the queries as follows:

A. `low OR fat OR goat OR feta OR swiss OR blue OR havarti OR muenster OR cheese`

B. `sugar OR gingerbread OR (chocolate OR chip) OR oatmeal OR cookie OR recipes`

C. `baking OR (wheat OR rye OR sweet OR (sourdough OR (sour adj dough)) OR ('sticky buns') OR (date adj nut) OR (coffeecake OR (coffee adj cake)) OR bread)`

5.3.2.2 Defining the Thesaurus in the Definition File

The second step in creating a thesaurus is adding a thesaurus statement to the appropriate database's definition file. Adding the thesaurus statement to a database's definition file enables the thesaurus operator for searching.

To define a thesaurus for a database, (only one thesaurus may be defined for any given database) add the following optional definition statement to the database's definition file (`.def`):

Format

```
thesaurus=PLWeb <thesname>
```

The following table explains the parameters you may use when defining a thesaurus in the `.def` file:

Table 5-1: Optional Definition File Statement: Thesaurus

Argument	Description	Explanation
`<thesname>`	Thesaurus File Name	Specifies the name of the thesaurus file you created under "Defining the Thesaurus File" on page 66.

Example

```
thesaurus=PLWeb c:\usr\dbases\thesauri\fedthes.txt
```

Note

If you use a relative path in `<thesname>`, PLWeb Turbo will consider it relative to the database directory (`<dbdir>`).

Tip

You may refer to the same thesaurus file from multiple database definition files.

5.3.2.3 Using a Thesaurus

End users have access to a database's thesaurus through use of the *thesaurus operator* (@). When the thesaurus operator is appended to a query term, PLWeb Turbo replaces that term with the predefined synonyms that are located in the thesaurus file.

✉✉ **Note** ✉✉

You can automatically invoke the thesaurus for all databases by editing string 227 in `defaults.cpl` (refer to "Activating Automatic Thesaurus Operation for All Query Terms" on page 238). However, if you turn on automatic thesaurus, it will not be effective in combination with operators that you apply explicitly. For example, if a user entered the query *run+*, a thesaurus search would not be performed. Only stemming would be performed.

Query Format
`<word>@`

Example
`FBI@`

Given the following thesaurus entry

```
FBI    federal bureau investigation FBI
```

and the query

```
FBI@
```

PLWeb Turbo would search on the terms *federal*, *bureau*, *investigation*, and *FBI*.

✉✉ **Note** ✉✉

If you use the thesaurus operator in conjunction with a word that is not a main entry in the thesaurus, the operator will be disregarded (e.g., if *osculate* is not a thesaurus entry, *osculate@* will be interpreted as *osculate*).

5.3.3 Modifying the Display of Field Labels in Records

By default, field labels will be displayed in PLWeb Turbo records that originate as ASCII documents. You can override this default for a particular database and suppress the display of field labels for ASCII records by adding the following optional definition statement to the database's definition file:

Format
`hide_fields=Y`

Y must be capitalized.

5.3.4 Optimizing I/O

You can optimize your system's I/O by storing parts of the index file in different locations, or *partitions*. You might choose to specify different devices/drives to optimize disk utilization for space required and/or performance. For instance, you may choose to store certain index partitions locally rather than on a networked drive in order to improve search and display performance.

5.3.4.1 Creating a Database Index Partition

By default, database index information is unified in a single `.pls` file. Dividing a database index into separate files is called *partitioning*. You can partition two components of a database index—the dictionary and storage keys—and save them on a faster storage medium.

The first component that you may partition is the *dictionary*, which contains an alphabetical list of all words indexed in the database and a posting count (number of occurrences) for each word. The second component consists of the database's *storage keys*, which lists the location and document class information for each record in the database.

Creating a Database Dictionary Partition

If you are creating a database on slower media, you can improve search speed by partitioning the database's dictionary to a faster medium. The size of this partition is approximately 20 times (in bytes) the number of unique words in the database. To create a dictionary partition for a database, you must add the following optional definition statement to the database's definition file:

Format

```
partition=<partname> dic
```

The following table explains the parameters you may use when defining a dictionary partition in the `.def` file:

Table 5-1: Optional Definition File Statement: Dictionary Partition

Argument	Description	Explanation
`<partname>`	Name of Partition File	Specifies the path and name of the dictionary partition file. If you use a relative path, it will be considered relative to the database directory (`<dbdir>`).

Example

```
partition=<drive>:\<installDir>\lawdbdic\acctlaw.dic dic
```

Caution

If you define a dictionary partition for an existing database, you must recreate or reorganize the database index.

Creating a Storage Keys Partition

If you are creating a database on slower media, you can improve the speed at which records and the hitlist are displayed by partitioning the database's storage keys to a faster medium. The size of this partition is approximately 12 times (in bytes) the number of records in the database. To create a stor-

age keys partition for a database, you must add the following optional definition statement to the database's definition file:

Format
```
partition=<partname> storage_keys
```

The following table explains the parameters you may use when defining a storage keys partition in the `.def` file:

Table 5-2: Optional Definition File Statement: Storage Keys Partition

Argument	Description	Explanation
<partname>	Name of Partition File	Specifies the path and name of the storage keys partition file. If you use a relative path, it will be considered relative to the database directory (<dbdir>).

Example
```
partition=<drive>:\<installDir>\lawdbstor\acctlaw.stk storage_keys
```

Caution

If you define a storage keys partition for an existing database, you must recreate the database index.

5.3.4.2 Combining Partitions

If you are partitioning both the dictionary and storage keys for a database, you can store the partitions in separate files, as discussed in the preceding sections, or you can combine them in a single file by adding the following optional definition statement to the database's definition file:

Format
```
partition=<partname> storage_keys,dic
```

The following table explains the parameters you may use when defining a combined dictionary and storage keys partition in the `.def` file:

Table 5-1: Optional Definition File Statement: Combined Dictionary/Storage Keys Partition

Argument	Description	Explanation
<partname>	Name of Partition File	Specifies the path and name of the dictionary and storage keys partition file. If you use a relative path, it will be considered relative to the database directory (<dbdir>).

Example
```
partition=<drive>:\<installDir>\lawdbdicstor\acctlaw.par storage_keys,dic
```

5.3.5 Controlling Index Size

At database creation time, PLWeb Turbo allows you to control how much information will be in the database index (`.pls` file). The more information in the index, the larger the index will be, requiring more storage resources. A smaller index requires less storage and may even increase PLWeb Turbo's query processing speed. However, more information in the index enables PLWeb Turbo to support more query operators. Weigh the advantages and disadvantages of the available index sizes before creating your database, and choose the index size that best suits your needs.

By default, index size is set to STANDARD. You can override the default for a particular database by adding the following statement to its definition file:

Format

```
index_size=<size>
```

The following table explains the parameters you may use for `<size>` when defining index size in the `.def` file. If you change the index size, you will need to recreate the database as described under "Creating a Database" on page 91.

Index size options are described below, in descending order of the size index they generate:

Table 5-2: Optional Definition File Statement: Index Size

Size	Explanation
FULL	A database with a FULL index contains the most information possible about the source text of the database and supports all of the operators in PLWeb Turbo's query language, including information about paragraph delimiters. PLWeb Turbo regards two consecutive line return characters in source text as a paragraph delimiter.
STANDARD	A database with a STANDARD index does not maintain information about paragraph delimiters. PLWeb Turbo will treat the entire contents of each field as a single paragraph; this means that the behavior of the *same* and *notsame* query operators will be undefined.
FIELDONLY	A database with a FIELDONLY index does not maintain information about paragraph delimiters (refer to FULL) or adjacency information for the terms of the database. This means that the behavior of the *same, notsame, adjacency, within, near,* and *at least* query operators will be undefined.
TINY	A database with a TINY index does not maintain information about paragraph delimiters, adjacency (refer to FIELDONLY), or fields. This means that the behavior of the *same, notsame, adjacency, within, near, at least,* and *field-restriction* operators will be undefined. PLWeb Turbo will also produce unpredictable results if it attempts to evaluate a query that contains any fieldlists that you may have established in your database definition file.

✉✉ Note ✉✉

Information on the operators mentioned in the above section may be found under "Table of Operators" on page 230.

5.4 Adding Comments

You may occasionally wish to add commentary to a definition file or temporarily disable individual definition statements. By preceding lines in a definition file with the pound sign (#) and a space, you can dictate those lines PLWeb Turbo will ignore.

Format

```
# <string>
```

The following table explains the variable for defining comments:

Table 5-3: Optional Definition File Statement: Internal Commentary

Argument	Description	Explanation
<string>	Commentary Text	Specifies your commentary text.

- <string> is your commentary text.

✉✉ **Note** ✉✉

Field names/attributes must precede optional definition statements and comments; however, comments and optional definition statements may be in any order.

---

6. Setting Global Application Parameters

The `plweb.conf` file contains database-wide configuration settings. Only one `plweb.conf` file exists per installation of PLWeb Turbo.

6.1 Chapter Checklist

Use this checklist to ensure that you have performed the necessary steps when configuring your `plweb.conf` file.

1. Edit the `[Databases]` section to include the names and paths of all databases—refer to the section below.

2. Edit the `[Views]` section to include the paths of all views—refer to page 78.

3. Edit the `[Configuration]` section to specify global configuration parameters—refer to page 78.

6.2 Configuration File Contents

The `plweb.conf` file contains three sections:

Databases Specifies the global list of databases.

Views Specifies the list and location of the different views.

Configuration Contains global configuration settings.

Format
The format of the `plweb.conf` file is as follows. Comment lines are allowed and are identified by a # character at the start of the line. Empty lines are also allowed (and ignored).

```
[Databases]
<localdbspec>
<virtualdbspec>

[Views]
<view_path_1>
<view_path_2>

[Configuration]
<key1=value1>
<key2=value2>
```

✉✉ **Note** ✉✉

Errors are logged to `<drive>:\<installDir>\plweb\logs\error.log`.

6.3 Databases Section

Database specifications in the [Databases] section can take one of two formats: one for physical databases and one for virtual databases. Physical databases **must be listed before** any virtual databases of which they are a part.

Databases are specified in the plweb.conf file as follows.

Format

```
name=<dbname>, location=<dbpath>, altdef=<defpath>, description=<dbdesc>
class=virtual, name=<dbname>, databases=<dbset>, description=<dbdesc>
```

The following table explains the keys and values for database specification:

Table 6-1: Database Specification: plweb.conf

Key	Value Description
name	Specifies the database name. The database name must begin with an alphabetic character and may contain alphanumeric characters, the underscore, and forward slash. For virtual databases, the predefined setting ALL specifies that all physical databases listed should be included in the virtual database.
location	Specifies the full path to database index file (.pls), including the name of the index file, but without the extension.
altdef	Specifies an alternate path to the database definition file (.def), if the file is either (a) not in the same location as the .pls file, and/or (b) has a different name (before the extension) than the .pls file.
description	Specifies a database description. The description cannot contain the comma character. In addition, the description can be accessed via the {$DBDESC} macro, as discussed on page 194.
class	Specifies that a database is virtual. virtual is a predefined argument that must be present for virtual databases.
databases	Specifies the set of databases that make up the virtual database, in the format of <dbname1>+<dbname2>+<dbname3>... The order of <dbset> is significant; it must reflect the same order as the list of physical databases. Virtual databases may be nested. Again, order is significant. For example, if a virtual database named db2 contains the virtual database db1, db1 must be defined as a virtual database before db2.

Example 1

The following example defines

- The physical databases classical, rock, blues, and jazz.
- The virtual database all, which includes all of the physical databases.

```
name=classical, location=c:\home\julie\musicdb\classical, description=Classi-
cal Music
name=rock, location=c:\home\julie\musicdb\rock, description=Rock Music
name=blues, location=c:\home\julie\musicdb\blues, description=Blues Music
name=jazz, location=c:\home\julie\musicdb\jazz, description=Jazz Music
class=virtual, name=all, databases=classical+rock+blues+jazz, descrip-
tion=Music DB
```


Example 2
The following example defines

- The physical databases `classical`, `rock`, `blues`, `jazz`, and `new_age`.
- The virtual database `music1`, which includes the `classical` and `rock` physical databases.
- The virtual database `music2`, which includes the `blues`, `jazz`, `classical`, and `rock` databases.

```
name=classical, location=c:\home\julie\musicdb\classical,description=Classical
Music
name=rock, location=c:\home\julie\musicdb\rock, description=Rock Music
name=blues, location=c:\home\julie\musicdb\blues, description=Blues Music
name=jazz, location=c:\home\julie\musicdb\jazz, description=Jazz Music
name=new_age, location=c:\home\julie/musicdb\newage, description=New Age Music
class=virtual, name=music1, databases=classical+rock, description=Class./Rock
Music
class=virtual, name=music2, databases=blues+jazz+music1, description=All Music
```

6.4 Views Section

Views are specified in the `plweb.conf` file as follows.

Format
`<viewpath>`

The following table explains the argument for view specification:

Table 6-2: View Specification: plweb.conf

Key	Explanation
`<viewpath>`	Specifies the path to a view. This path must be absolute. For information on creating views, refer to “Creating a View” on page 82).

Example
`c:\home\julie\plweb\views\music`

6.5 Configuration Section

The `[Configuration]` setting allows you to specify global application parameters. The number of configuration settings that is supported here is limited, since most are set at the view level to allow greater customizability.

Currently, only two global configuration parameters are supported: the temporary directory used by PLWeb Turbo, and the file formats for which PLWeb Turbo should support the `SRCLINK` macro.

Format
<key=value>

The following table explains the key and value pairs for the [Configuration] section of the plweb.conf file.

Table 6-3: View Specification: plweb.conf

Key	Value Description
<TempDirectory>	Specifies the path to the temporary directory PLWeb Turbo should use.
<SrcLinkFormats>	Specifies for which record formats PLWeb Turbo should allow SRCLINK macros. If a SRCLINK macro is specified for a document, PLWeb Turbo will determine the record format (reader) of the document. If the format is listed in this setting, a link to the document will be generated. If the format is not listed, no link will be generated. The formats are separated by commas. Valid formats are as follows: CplPlsHtmlReader CplAsciiRecord CplStddoc CplAcrobatReader For more information on the SRCLINK macro, refer to page 222.

Example
```
[Configuration]
TempDirectory=c:\tmp
SrcLinkFormats=CplAcrobatReader, CplOutsideInReader
```


7. Creating and Configuring Views

A view is the key structural element for all database administration tasks. You can configure views by altering the following files, which are located in the view directory (<drive>:\<installDir>\plweb\views\<viewname> by default):

Table 7-1: View-level Configuration Files

File/Directory	Description
view.conf	Controls most of the view configuration settings, such as view name, view description, accessible databases, logging, relative link resolution, and the default operator.
viewusrs.tab	Contains a list of users who have access to the view.

Additionally, you can configure the user interface in the <drive>:\<installDir>\plweb\view\<viewname>\templates directory.

7.1 Chapter Checklist

Use this checklist to ensure that you have performed the necessary steps when creating and configuring views.

1. Create a view—refer to the section below.
2. Configure the view.conf file—refer to page 83.
3. (optional) Configure the viewusrs.tab file—refer to page 86.

7.2 Creating a View

1. Create a subdirectory called <viewname> in the <drive>:\<installDir>\plweb\views\ directory, replacing <viewname> with the name of the view you are creating. You must have at least one view per PLWeb Turbo installation.

✉✉ **Note** ✉✉

The default view selection screen template is implemented such that, when users have access to only one view, they will not see the view selection screen; they will go directly to the search screen. You may change this at any time by removing the first three lines of the view.tmpl template.

Note

The <viewname> directory does not have to be under the plweb directory structure. It can be in any location on any physical machine, as long as it is referenced correctly in plweb.conf. For more information on plweb.conf, refer to Chapter 6.

2. Create and configure a view.conf file in the <viewname> subdirectory using the guidelines under "Configuring the view.conf File" on page 83.

3. Add the path to the <viewname> directory to the plweb.conf file using the guidelines under "Views Section" on page 78.

4. If you plan to enable access restrictions, create a viewusrs.tab file in the <viewname> subdirectory using the guidelines under "Configuring the viewusrs.tab File" on page 86.

5. If you intend to create view-specific templates for one or more views, create a templates subdirectory under the <drive>:\<installDir>\plweb\views\<viewname> directory, which is where you will store templates pertinent to that view. If you do not create view-specific templates, PLWeb Turbo will default to those listed in <drive>:\<installDir>\plweb\templates. For more information on configuring templates, refer to Chapter 11.

Tip

It is often easiest to configure the view.conf and viewusrs.tab files using existing ones, such as those in the <drive>:\<installDir>\plweb\views\sample-view, as models. For more information on the default templates that ship with PLWeb Turbo, refer to Appendix B.

7.3 Configuring the view.conf File

The view.conf file determines view-specific settings for each database. The file is a simple collection of key value pairs, one pair per line. Comments are allowed by using the # as the first symbol on the line. Keys are not case sensitive.

The following table describes the keywords and values you can use in the view.conf file.

Table 7-2: Keywords and Values: view.conf

Keyword	Description	Key and Value Example
name	Specifies the view name. The name of the view cannot contain spaces.	Name = view1
Description	Specifies the view description. The view description must be contained to one line and may not include the equals sign.	Description = PLWeb Turbo sample view

Table 7-2: Keywords and Values: view.conf

Keyword	Description	Key and Value Example
`Attributes`	Specifies the view attributes. This value may be either `hidden` or `none`. `None` indicates no attributes. `Hidden` indicates that the user cannot see the view.	`Attributes = NONE`
`Databases`	Specifies the list of databases to be accessible from this view. When listing multiple databases, separate each database by a comma.	`Databases = Travel, Help, News`
`Logging`	Specifies whether logging is `on` or `off`.	`Logging=off`
`Logfiles`	Lists all log files that need to log activities in this view. Separate multiple log files by commas. For each log file, supply a logical name and a location in the following format: `logical_name:location`. The location is an absolute or relative path to the log file. If it is relative, it is assumed to be relative to the view directory. Log files are referred to in templates by their symbolic names only for security reasons.	`Logfiles = querylog:logs\query.log, doclog:logs\doc.log`
`ModifyRelativeLinks`	Directs PLWeb Turbo to turn on the modification of relative links. Relative links that are present in HTML documents retrieved from a database will not work without PLWeb Turbo adding markup to the document. Setting this value to `TRUE` will cause PLWeb Turbo to change all relative links to absolute links. If you want your relative links to remain relative, you can add the `BASEURL` macro (refer to page 192) to the `doc_html_impl.tmpl` template (refer to page 182). The `BASEURL` macro will add an HTML base-URL entry to the document. Set this value to `FALSE` if you use the `BASEURL` macro.	`ModifyRelativeLinks = TRUE`

Table 7-2: Keywords and Values: view.conf

Keyword	Description	Key and Value Example
`DefaultOperator`	Specifies the default search operator. This value (which may be either `OR`, `AND`, or `ADJ`) is used when interpreting the query rule (page 141), only if the query rule fails to use explicit operators to connect query components. If the query rule does contain search operators, they will override any operator stated for this value. It is recommended that you explicitly connect query components in the query rule rather than rely on this setting. There is a separate default operator for the query proper, which is set in the search form (page 172).	`DefaultOperator = OR`
`PreparseTemplates`	Allows you to turn off preparsing of templates on a view basis. Enables maximum performance for views being accessed by users while you modify and debug templates for a view under development, without having to restart the daemon for changes to those templates to take affect. This key and value pair is not a default of the `view.conf` file. Therefore, you must add it, as shown to the right, in order to turn off preparsing of templates. When you add this key and value pair to a view's `view.conf` file, preparsing will be turned off for all templates in the view except for the `views.tmpl` and `views_impl.tmpl` templates. This is because the view selection screen is not associated with a view. Using this statement in the `view.conf` file will impact performance. Therefore, we do not recommend using it on a live site. It is mainly meant to be used in a testing environment.	`PreparseTemplates = False`

Sample view.conf File

```
#  View name
name = view1

#  View description
description = PLWeb Turbo sample view

#  No view attributes
attributes = NONE

#  List of databases to be accessible from this view
databases = Travel, Help, News

#  No logging
logging = OFF
```



```
#  Define two log files: logs/query.log and logs/doc.log, referred
#  to in templates as querylog and doclog, respectively
logfiles = querylog:logs/query.log, doclog:logs/doc.log

#  Turn on the modification of relative links
ModifyRelativeLinks = TRUE

#  Default search operator
DefaultOperator = OR
```

7.4 Configuring the viewusrs.tab File

The `viewusrs.tab` file contains the list of users that have access to the view. When authentication is turned on, only those users listed in this file will have access to the view.

The format of this file is one user per line.

```
user1
user2
user3
```

PLWeb recognizes a special keyword, `_ALL_USERS_`, which, if entered as the first user in the `viewusrs.tab` file, allows all users access to the view. This enables administrators to create restricted and unrestricted views.

It is recommended that your `viewusrs.tab` file have the `_ALL_USERS_` entry as the first entry in the list if HTTP daemon authentication is turned off. This prevents access problems if the browser submits unnecessary authentication information to the daemon, which can happen if the administrator switches between having authentication turned on and off.

Example viewusrs.tab File with Restricted Access

```
hrmn8r
janed0
```

Example viewusrs.tab File without Restricted Access

```
_ALL_USERS_
```


8. Creating and Updating a Database

This chapter will guide you through building and maintaining your PLWeb Turbo databases. An important thing to know is that most database maintenance can occur while your users are actively searching and retrieving data. A database need not be restricted from users when you are adding, deleting, or updating files.

✉✉ **Note** ✉✉

PLWeb Turbo administration utilities discussed in this chapter log their errors to `<drive>:\<installDir>\plweb\logs\admin_err.log`.

8.1 Chapter Checklist

Use this checklist to ensure that you have performed the necessary steps when creating a database:

1. Check your environment variables—refer to page 91.
2. Create the database—refer to page 91.
3. Make the database visible to users—refer to page 92.
4. Add files:
    - Local files—refer to page 94.
    - Remote files—refer to page 95.
    - Binary files—refer to page 98.
    - HTML files that include relative links—refer to page 97.

When updating a database or performing database maintenance, you can perform the following operations:

- Delete records—refer to page 103.
- Delete source files—refer to page 104.
- Update records—refer to page 105.
- Synchronize a database—refer to page 107.
- Merge databases—refer to page 109.
- Reorganize the database index—refer to page 110.
- Delete a database—refer to page 112.

- Move a database—refer to page 113.
- Inspect a database and verify its integrity—refer to 100.
- Get database information—refer to 112.

✉✉ **Note** ✉✉

Instructions for using EZ Admin and PLSpider may be found in the online help for those applications.

When executing a PLWeb Turbo command-line utility, any relative paths passed will be considered relative to the directory from which you executed the utility.

In addition, keep in mind that any time you create, delete, or update a database, you may need to alter the `dbdef.html` file. The `dbdef.html` file, which may be found in `<drive>:\<installDir>\plweb\htdocs`, contains information about each database.

8.2 Checking the Environment Variables

Before executing any of the PLWeb Turbo command line utilities, check that your environment variables are set correctly. For more information, refer to “Checking the Environment Variables” on page 42.

8.3 Creating a Database

The first step in creating a PLWeb Turbo database is invoking the utility program `plcreate`. Before running this utility, you must create the database's definition file (`.def`) with a text editor. For further information on the database definition file, refer to “Creating the Database Definition File” on page 60.

Execute the `plcreate` utility from the command line as follows:

```
plcreate [-v] <dbname> [-f <dbpath>] [-d <description>]
```

The following table explains the command-line parameters for `plcreate`:

Table 8-1: Command-line Parameters: plcreate

Flag/Argument	Description	Explanation
`-v`	Verbose	Causes verbose output during processing.
`<dbname>`	Database Name	Specifies the name of the database to create. Also specifies the `.pls` file name if you do not use the `-f` option. Adds the database name to the `plweb.conf` file. `<Dbname>` must begin with an alphabetic character, can contain alphanumeric characters (with no spaces), and can be up to 35 characters in length.
`-f <dbpath>`	Path to Database Definition File	Specifies the directory path and name of the database's definition file (`.def`) without the file extension. You must use this argument to specify a path for the definition file (1) when you are not launching `plcreate` from the directory in which you are creating the database or (2) you are giving the `.pls` file a name other than `<dbname>`. If you do not include the argument, the index file name will be `<dbname>.pls`. If you attempt to create a database using a name in `<dbpath>` for which an index file (`.pls`) already exists in the same directory, the command will fail. It will also fail if `<dbname>` is already used in `plweb.conf`. To avoid a failed command, you must (1) delete the database with `plrmdb` (refer to page 112); and (2) re-execute the `plcreate` command.
`-d "<description>"`	Database Description	Specifies the description of the database. The database description cannot contain commas, single or double quotation marks, or angle brackets. In addition, the description must fit on one line. The `-d <description>` flag and argument plays an integral role with the `plweb.conf` file, as follows: (1) If the `-d` flag is specified with a non-empty description, the contents of the description will be used in the `description` field of the `plweb.conf` file. (2) If no `-d` option is specified, the description in `plweb.conf` will default to the short name of the database. (3) If the `-d` option is specified with an empty description (`plcreate -v xzy -d ""`), the description field in the `plweb.conf` file will be empty. In other words, you can override the behavior in (2), and force the field to remain empty.

Example

```
plcreate fedlaw
plcreate Federal_Law -f c:\usr\db\fedlaw
```

8.4 Displaying a Database

After running the `plcreate` utility, the `plweb.conf` file will contain all databases available for searching. It will not, however, automatically make those databases visible to end users. To make the databases searchable and visible to end users, you will need to

1. Add the database to the views in which you want to make them available by adding its `Databases` setting to the `view.conf` file (refer to page 83).

2. Make the database visible by using the `DBNAME` and `DBDESC` macros in your interface (refer to page 172 and page 195).

3. Stop and then start the PLWeb Turbo daemon as follows:

1. From the START button, select Control Panel.

2. In the Control Panel window, double click the SERVICES icon.

3. In the Services window, select the `plwebd` service and click STOP and then, click START.

   After the server opens the local databases (you will get notification messages of this event), PLWeb Turbo will respond to user requests.

8.5 What Happens During Indexing?

The process in which PLWeb Turbo adds, deletes, or updates database files and records is called *indexing*. The indexing process does not alter your source files in any way; it simply reads their contents and records its findings in a separate *index file* (`.pls`).

If you are adding new files to a database, PLWeb Turbo scans your source files (also known as container files, because each “contains” one or more records) and notes the number of occurrences and location of all words (except stopwords) in the database's index file. Conversely, when you delete files or records, PLWeb Turbo removes references to their constituent words from the index.

For each new record added to a database, PLWeb Turbo assigns a *record ID*, which indicates the order in which the record was added (e.g., the 99th document added to a database would be assigned document ID 99). Within a database, each document's ID number is unique.

8.6 Preparing to Index Files

Before adding files to a database, ensure that you have sufficient hard disk space for the temporary files that will be created during the indexing process. It is recommended that the amount of available space be equal to the amount of data being added (e.g., to index 1 MB of data, you should have 1 MB of free disk space). The resulting index will occupy approximately half of this space; the rest is used for creation of temporary files and will become free when indexing has completed.

You may specify the path to any source file to the administration utilities as either an absolute or relative path. The utilities will convert any absolute path to the equivalent relative path with respect to the location of the database index (`.pls`) file. This derivation allows the administrator the freedom to move a database and its source files, as long as the relative location of the source files to the database index remains the same. For more information, refer to “Moving a Database” on page 113.

Example
Suppose that the database index file is c:\usr2\travel\cruises.pls and the administration utility is executed from the database directory (c:\usr2\travel).

The following table illustrates how PLWeb Turbo would resolve each of the input arguments.

Table 8-2: Relative Path Resolution for Source Files

Input File Name Argument	Computed Path	Actual Absolute Path
cancun.txt	.\cancun.txt	c:\usr2\travel\cancun.txt
..\alaska.txt	..\alaska.txt	c:\usr2\alaska.txt
\xtra\qe2.txt	..\..\xtra\qe2.txt	c:\xtra\qe2.txt

8.7 Adding Files

8.7.1 Adding Local Files

With the pladd utility, you can add local files to a database that has been created with either the plcreate or plezdb utility. Files added with pladd must be of the following format: ASCII, HTML, or Adobe Acrobat.

✉✉ **Note** ✉✉

If your HTML files contain relative links, refer to page 97 for instructions on adding those files to your database.

Execute the pladd utility from the command line as follows:

```
pladd [-v] [-a <appendfile>] [-f <listfile>] <dbname> [<addfile> {<addfile>}]
```

The following table explains the command-line parameters for `pladd`:

Table 8-3: Command-line Parameters: pladd

Flag/Argument	Description	Explanation
`-v`	Verbose	Causes verbose output during processing.
`-a <appendfile>`	Append to File	(ASCII and HTML only.) Specifies an existing database source file to which records contained in `<addfile>` should be appended. HTML files in `<addfile>` must include `<!--plsfield:end-->` record tag markup at the end of records. PLWeb Turbo does not need the original source files that you are adding once indexing is completed; therefore, you can delete or move them. If you do not wish to append files to an existing container, do not specify this argument. The files you index will become part of the database and therefore, cannot be deleted or moved relative to the database.
`-f <listfile>`	List of Files to Add	Directs `pladd` to add files listed in `<listfile>`. You must create `<listfile>` yourself. `<listfile>` must be a text file and must list the paths and names of the files to be added. When creating `<listfile>`, observe the following guidelines: (1) one or more file names should be listed; (2) file names must be separated by a new line; and (3) each file name may include a path. If you use the `-f` flag, `<addfile>` arguments are not required. If you use both the `-f` flag and the `<addfile>` argument, the files specified by `<addfile>` will be indexed first.
`<dbname>`	Database Name	Specifies the name of the database to which you are adding files.
`<addfile>`	File to Index	Specifies the path and name of the file being indexed. This may be a single file or a list of files.[a]

a. If you have multiple files to index, instead of running `plcreate` on each individual file, create a file that lists those files and then index that file list. For example, to create a file containing a list of all the `.txt` files in the current directory, execute `dir /B *.txt > file.lst`. Then, run `pladd` with `file.lst`, as the argument for the `-f` option (`pladd dbname -f file.lst`) as described above.

Examples

```
pladd fedlaw c:\usr\db\fl.001
pladd -a c:\usr\db\fl.001 fedlaw c:\usr\db\fl.002
pladd -f c:\usr\db\lawdocs.lst fedlaw
```

Tip

When adding multiple files, we recommend that you execute the `pladd` command once for all files rather than once per individual file. This will be significantly faster.

8.7.2 Adding Remote Files

A file is *remote* if it resides in a file system on another machine that is not mounted on the machine running PLWeb Turbo and is accessible by an HTML client. You can add a remote file to a PLWeb Turbo database with the `plremote` utility. The utility retrieves the remote document and indexes it.

After indexing the text, `plremote` generates a local surrogate file, which contains a URL to the original document and summary information. If the original document is in either ASCII or HTML format, the local copy is then discarded.

A remote file to be added with `plremote` may be in ASCII, HTML, or Adobe Acrobat format. The file must also be accessible by a URL. In addition, a remote file cannot be subdivided into multiple records; however, remote ASCII and HTML files can contain field markup. (For more information on field markup, refer to “Adding Field Markup” on page 49.)

When indexing remote ASCII or HTML files, a file containing a summary of the document contents is placed in the surrogate directory. When accessing title or summary information (using the `TITLE` and `SUMMARY` macros, which are described on pages 224 and 223), this file is accessed without re-retrieving the document. When the document itself is requested (using the `DOC` macro, page 197), it is re-retrieved, so that the document shown is always current.

✉✉ **Note** ✉✉

HTML files that contain relative links require special treatment. If your HTML files contain relative links, you will need to add those files to your database using the instructions described in Chapter 9.

When indexing remote files other than ASCII or HTML, it is not possible to store the summaries. The `plremote` utility stores the full content of the document instead, and the document becomes a local file. When the document content is requested by PLWeb Turbo (using the `DOC` macro, page 197), it is retrieved from the local file, and is never re-retrieved.

✉✉ **Note** ✉✉

The `RMTLINK` macro (described on page 218) is not supported for remote Acrobat files.

Execute the `plremote` utility from the command line as follows:

```
plremote [-v] [-n] [-o] [-c] [-f <filelist>] [-p <surpath>] <dbname> [<url
{url}>]
```

Table 8-4: Command-line Parameters: plremote

Flag/Variable	Description	Explanation
`-v`	Verbose	Causes verbose output during processing.
`-n`	No Update	Specifies that `plremote` should fetch the URLs and store them locally but not update the database.
`-o`	Keep Original	Specifies that the local files that store the contents of the URL should not be replaced by the surrogate contents.
`-c`	Collapse	Specifies that all surrogate files should be stored in a single directory. If this option is not specified, `plremote` will create a directory tree to store the surrogate files.

Table 8-4: Command-line Parameters: plremote

Flag/Variable	Description	Explanation
`-f <filelist>`	File List	Specifies a list from which `plremote` should read the URLs to be added to the database.
`-p <surpath>`	Surrogate Path	Specifies the path to the directory in which to store the surrogate files. If this option is not specified, `plremote` will use the "surrogate" subdirectory of the database directory.
`<dbname>`	Database Name	Specifies the name of the database to which you are adding remote files.
`<url>`	URL to Add	Specifies the URL to be added. The URL can optionally include the `http://` protocol specification, which is the only protocol supported by PLWeb Turbo.

Examples

```
plremote UrlDb http://www.ibm.com
plremote -v UrlDb www.ibm.com www.microsoft.com
plremote -v -f url.lst UrlDb
plremote -v -o -f url.lst UrlDb
```

8.7.2.1 Where plremote Generates Surrogate Files

By default, `plremote` creates a directory tree under the surrogate directory, causing corresponding directories to be generated. For example, the URL

```
http://www.abc.com/products/rs6000.html
```

will cause a surrogate file to be generated in

```
surrogate\www.abc.com\products\rs6000.html
```

Alternatively, `plremote` can store all the surrogate files directly in the surrogate directory. If used, the `-c` flag will collapse the URL into a flat file name so the URL can be stored in the surrogate directory. For example, if the `-c` option was specified, the URL above would cause the following surrogate file to be generated:

```
surrogate\abc.products.rs6000.html
```

8.7.3 Adding HTML Files that Contain Relative Links

You can add and update HTML files that contain relative links just like you would any other HTML or non-HTML files. You simply use `pladd`, `plremote`, and `plsync`. Depending on the source files and their location, however, you will need to make some changes to your configuration and/or templates in order for PLWeb Turbo to resolve the relative links when your users retrieve these files. Because adding HTML files that contain relative links to a database presents a special case, we have devoted an entire chapter to the process you need to follow. For further information on this process, refer to Chapter 9.

8.7.4 Adding Binary Files

If you wish to add a binary file (e.g., `.wav`) to a database but the file type is not supported by PLWeb Turbo (i.e., it is not listed under "List of Document Formats and Filters" on page 232), you can add the file with the `pladdsur.pl` utility.

The `pladdsur.pl` utility is commonly used to add files to a database of images. For each image you have, you would create a corresponding text file that describes the image. Then, when a search is performed on the indexed text (not the image), if the user chooses to display the document, the URL to the binary file is referenced. It is then up to the user's Web browser to handle the binary file properly (i.e., display the image). The image files remain at their original location.

Before adding a remote binary file with `pladdsur.pl`, you must have a local copy of the text file that describes the binary file and know the URL to the original binary file.

✉✉ **Note** ✉✉

The `pladdsur.pl` file must be executed from `<drive>:\<installDir>\plweb\bin`.

After adding a text file with `pladdsur.pl`, you can delete the local copy of the text file.

Execute the `pladdsur.pl` utility from the command line as follows:

```
perl pladdsur.pl [-v] [-i] <dbname> <URL> <headline> <surrogate_file>
[<lcpyRemfile>]
```

or

```
perl pladdsur.pl [-v] [-f <listfile>] <dbname>
```

The following table explains the command-line parameters for adding binary files with `pladdsur.pl`.

Table 8-1: Command-line Parameters: pladdsur.pl.

Flag/Argument	Description	Explanation
`-v`	Verbose	Causes verbose output during processing.
`-i`	Read from Standard In	Directs `pladdsur.pl` to treat standard in as the text of the remote file. If you use the `-i` option, you must not specify the `<lcpyRemfile>` argument.

Table 8-1: Command-line Parameters: pladdsur.pl.

Flag/Argument	Description	Explanation
`-f <listfile>`	List of Files to Add	Directs `pladdsur.pl` to add files listed in `<listfile>`. You must create `<listfile>` yourself. `<listfile>` must be a text file and must list the paths and names of the files to add. When creating the `<listfile>`, observe the following guidelines: (1) each entry must include the following arguments in the specified order; these arguments are explained later in this table: `<surrogate_file> <copyRemfile> <URL> <headline>` (2) the above arguments must be separated by spaces or tabs; and (3) each entry is separated by a new line. If you use this option, no other arguments after `<dbname>` are required.
`<dbname>`	Database Name	Specifies the name of the database to which you are adding files.
`<URL>`	Document URL	Specifies the full, valid URL of the remote document as it exists on its host machine.
`<headline>`	Hitlist Summary	Specifies the text to be displayed in the hitlist. `<headline>` can be one word or a string of words enclosed in quotation marks; however, in the actual `<listfile>`, the string *may not* be enclosed in quotation marks. `<headline>` may not contain HTML markup.
`<surrogate_file>`	Name of Surrogate File	Specifies the path (optional) and name of the file that will permanently reside on the local drive as part of the database. This file contains the location (URL) of the remote document. The file name for `<surrogate_file>` cannot be the same as the file name for the local copy of the text file being indexed. This file will be generated automatically during indexing.
`<lcpyRemfile>`	Local Copy of Remote File	Specifies the local copy of the text file you wish to add. The name may include a path. If you specify the `<lcpyRemfile>` argument, you must not use the `-i` option.

Example

```
perl pladdsur.pl paintings\
    http://www.abc.com/paintings/louvre.html "Welcome to the Louvre"\
    louvre.sur louvre.html
```

List Example

Given the following `pladdsur.pl` command

```
perl pladdsur.pl -f images.txt Louvre
```

the `<listfile>`, `art.txt`, might look as follows:

```
ml.sur ml.html http://www.abc.com/images/mona.gif Mona Lisa
sf.sur sf.html http://www.abc.com/images/sunflowers.gif Sunflowers
lr.sur lr.html http://www.abc.com/images/reves.gif Les Reves
```


8.8 Verifying Database Integrity

The `plverify` utility enables you to verify the integrity of a database. By default, `plverify` will perform a basic verification, which verifies that the dictionary is in order, that the count for each dictionary entry is reasonable, and that dictionary postings pointers are (physically) in bounds. You can increase the levels of verification to intermediate or advanced by specifying the appropriate option, as explained in the following table.

Execute the `plverify` utility from the command line as follows:

```
plverify [-v] [-{ai}] <dbname>
```

The following table explains the command-line parameters for verifying a database's integrity with `plverify`.

Table 8-2: Command-line Parameters: plverify

Flag/Argument	Description	Explanation
`-v`	Verbose	Causes verbose output during processing.
`-i`	Intermediate Verification	Verifies that the dictionary is in order, that the count for each dictionary entry is reasonable, and that dictionary postings pointers are (physically) in bounds In addition, verifies that the posting list associated with each dictionary term is in ascending order with (physically) correct record identifiers. Does not detect pointers to deleted records. Note that this will take longer to run than basic verification.
`-a`	Advanced Verification	Executes the same verification as the intermediate verification level (above) and detects deleted records and bad file IDs. Note that this can take a long time on large databases.
`<dbname>`	Database Name	Specifies the name of the database you wish to verify.

Example

```
plverify -v -a Travel
```

Given the above example you would receive the following output if no problems were found.

```
Starting the advanced verification of database Travel.
Database Travel verified, no problem discovered.
```

8.9 Accessing Database and Document Information

The `pldbinfo` utility allows easy access to database and document information. In particular, the utility allows the administrator to retrieve:

- General database information.
- The list of containers that are part of a database.
- The list of fields.

- General document information.
- The contents of a particular document.

Execute the `pldbinfo` utility from the command line as follows (you must specify at least one option):

```
pldbinfo [-i] [-c] [-f] [-l <docid> [-n <numdocs>]] [-r <docid>] [-d <docid>]
<dbname>
```

The following table explains the command-line parameters for accessing database and document information with `pldbinfo`.

Table 8-3: Command-line Parameters: pldbinfo

Flag/Argument	Description	Explanation
`-i`	Database Information	Directs `pldbinfo` to display general information for this database, including CPL version, database name, database location, number of unique containers, and number of unique terms.
`-c`	Database Containers	Directs `pldbinfo` to display the names of containers that are part of the database.
`-f`	Database Fields	Directs `pldbinfo` to display the list of fields and field attributes defined for the database.
`-l <docid>`	Database Document List	Directs `pldbinfo` to display a list of the documents in the database. For each document listed, it displays document ID and size, reader or file format, and container name. The display starts at the document ID specified by `<docid>` and continues for 1024 documents unless the end of the database is reached first or the `-n` option is specified.
`-n <numdocs>`	Number of Documents to List	Used only with the `-l` option, specifies the number of documents to be listed when using the `-l` option. The default value is 1024.
`-r <docid>`	Record Information	Directs `pldbinfo` to display general information for the given record, including record class name, record format, and container name.
`-d <docid>`	Record Display	Directs `pldbinfo` to display the contents of the given record.
`<dbname>`	Database Name	Specifies the name of the database.

Examples

```
pldbinfo -l 0 -n 512 Help
-- DATABASE DOCUMENTS --
ID     Size   Reader/Format         Container
----   -----  -----------------     ----------------------
 1     2119   CplPlsHTMLReader      htmlsrc\about.html
 2     1385   CplPlsHTMLReader      htmlsrc\advexpl.html
 3     3372   CplPlsHTMLReader      htmlsrc\advtool.html
 4     3619   CplPlsHTMLReader      htmlsrc\boolops.html
 5     2483   CplPlsHTMLReader      htmlsrc\compqry.html
 6     1589   CplPlsHTMLReader      htmlsrc\conop.html
 7     1619   CplPlsHTMLReader      htmlsrc\consrch.html
 8     2195   CplPlsHTMLReader      htmlsrc\dbfund.html
 9     1234   CplPlsHTMLReader      htmlsrc\defop.html
10     1450   CplPlsHTMLReader      htmlsrc\dicadv.html
```



```
11      1922    CplPlsHTMLReader      htmlsrc\fld1op.html
12      1609    CplPlsHTMLReader      htmlsrc\fld2op.html
13      781     CplPlsHTMLReader      htmlsrc\fldops.html
14      1414    CplPlsHTMLReader      htmlsrc\fuzadv.html
15      1926    CplPlsHTMLReader      htmlsrc\fuzop.html
16      1481    CplPlsHTMLReader      htmlsrc\intsrch.html
17      930     CplPlsHTMLReader      htmlsrc\natlang.html
18      4490    CplPlsHTMLReader      htmlsrc\nearops.html
19      2502    CplPlsHTMLReader      htmlsrc\oltoc.html
20      2987    CplPlsHTMLReader      htmlsrc\opexpl.html
21      1119    CplPlsHTMLReader      htmlsrc\precrul.html
22      1098    CplPlsHTMLReader      htmlsrc\ptfldnm.html
23      1648    CplPlsHTMLReader      htmlsrc\reladv.html
24      2516    CplPlsHTMLReader      htmlsrc\relrank.html
25      1969    CplPlsHTMLReader      htmlsrc\resscrn.html
26      2422    CplPlsHTMLReader      htmlsrc\schscrn.html
27      1377    CplPlsHTMLReader      htmlsrc\srchweb.html
28      4538    CplPlsHTMLReader      htmlsrc\valops.html
29      1715    CplPlsHTMLReader      htmlsrc\viewrec.html
30      1383    CplPlsHTMLReader      htmlsrc\virtdb.html
31      3107    CplPlsHTMLReader      htmlsrc\wildops.html
32      1749    CplPlsHTMLReader      htmlsrc\xphrop.html
33      1064    CplPlsHTMLReader      htmlsrc\xwdop.html
```

pldbinfo -i Travel

```
-- DATABASE INFO --
CPL version:                    CPL(tm)   6.3.25.0.483.483.0 Apr 17 1997
17:27:54
Database name:                  Travel
Database full name:             \home\robin\plweb\databases\travel\travel
Number of unique containers:    1
Number of unique terms:         10847
Number of records:              988
Highest record ID:              988
Database index is stemmed:      FALSE
Query-time stemming:            TRUE
Compacted database:             FALSE
Database time stamp:            Thu Sep 25 15:44:57 1997
```

pldbinfo -r 323 Travel

```
-- RECORD INFO --
Class name:             CplStddoc
Name of record format:  PL Standard Format
Container name:         travel.src
Record offset:          371056
Record size:            526
Class key size:         10
Application key size:   19
Container key size:     11
Record key size:        8
```

```
pldbinfo -c News
-- DATABASE CONTAINERS --
c:\home\robin\plweb\databases\news\news1.src
c:\home\robin\plweb\databases\news\news2.src
c:\home\robin\plweb\databases\news\news3.src
c:\home\robin\plweb\databases\news\news4.src
c:\home\robin\plweb\databases\news\news5.src
c:\home\robin\plweb\databases\news\news6.src
c:\home\robin\plweb\databases\news\news7.src
c:\home\robin\plweb\databases\news\news8.src

pldbinfo -f News
-- DATABASE FIELDS --
headline : DST
sub_headline : DST
text : DST
byline : DS
dates : DS
dateline : DS
```

8.10 Deleting Records

You can delete records by document ID from a database with the `pldelete` utility.

Execute the `pldelete` utility from the command line as follows:

```
pldelete [-v] [-f <listfile>] <dbname> <docID> [{<docID>}]
```

The following table explains the command-line parameters for `pldelete`:

Table 8-4: Command-line Parameters: pldelete

Flag/Argument	Description	Explanation
`-v`	Verbose	Causes verbose output during processing.
`-f <listfile>`	List of Records to Delete	Directs `pldelete` to delete records listed in the `<listfile>` text file. The `<listfile>` argument specifies the path and name of the text file in which ID numbers of records to be deleted are listed. When creating `<listfile>`, observe the following guidelines: (1) list one or more record IDs and (2) separate ID numbers with a new line.
`<dbname>`	Database Name	Specifies the name of the database from which records are to be removed.
`<docID>`	IDs of Documents to Delete	Specifies the ID number of each document to be deleted. This may be a single item or a list separated by spaces.

Example 1
In this example, records with IDs 98, 99, and 143 will be deleted from the `fedlaw` database.

```
pldelete fedlaw 98 99 143
```


Example 2
Example two accomplishes the same thing as example one, but uses the listfile `recs2del.txt`.

```
pldelete -f c:\usr\db\recs2del.txt fedlaw
```

where

- `recs2del.txt` contains the following record IDs:

```
98
99
143
```

Tip
When deleting multiple records, we recommend that you execute the `pldelete` command once for all records rather than once per individual record. This will be significantly faster.

8.11 Deleting Source Files

You can delete source files by file name from a database with the `plrmfile` utility.

Execute the `plrmfile` utility from the command line as follows:

```
plrmfile [-v] [-f <listfile>] <dbname> [<delfile> {<delfile>}]
```

The following table explains the command-line parameters for `plrmfile`:

Table 8-5: Command-line Parameters: plrmfile

Flag/Argument	Description	Explanation
`-v`	Verbose	Causes verbose output during processing.
`-f <listfile>`	List of Files to Delete	Directs `plrmfile` to delete source files listed in the `<listfile>` text file. The `<listfile>` argument specifies the path and name of the source files to be deleted. When creating `<listfile>`, observe the following guidelines: (1) list one or more source files and (2) separate source files with a new line.
`<dbname>`	Database Name	Specifies the name of the database from which source files are to be removed.
`<delfile>`	File to Delete	Specifies the path and name of each source to be deleted. This may be a single item or a list separated by spaces.

Note

Remember, when removing remote source files with `plrmfile`, you must specify the path to the surrogate file as well as the surrogate file name.

Example 1
In this example, the source files `statutes.html`, `laws_95.txt`, `regs.txt`, and `table1.pdf` will be deleted from the `fedlaw` database.

```
plrmfile fedlaw statutes.html laws_95.txt regs.txt table1.pdf
```

Example 2
Example two accomplishes the same thing as example one, but uses the listfile `sourcerm.txt`.

```
plrmfile -f c:\usr\db\sourcerm.txt fedlaw
```

where

- `sourcerm.txt` contains the following file names:

```
statutes.html
laws_95.txt
c:\usr\db\regs.txt
..\table1.pdf
```

Tip

When deleting multiple source files, we recommend that you execute the `plrmfile` command once for all source files rather than once per individual source file. This will be significantly faster.

8.12 Updating Records

You can update a record in an existing database by using the `plupdate` utility to replace it with the contents of a specified file.

One reason you might need to update records is, for example, if you wanted to add a new field and its associated text to "old" source files in an existing database. In this case, after having added the field to your database definition file (refer to "Adding a New Field to an Existing Definition File" on page 62), you would need to

1. Make a copy of each source file to which you will be adding the new field—do not modify the original.

2. Add the new field and its associated text to the copies.

3. Run the `plupdate` utility on the copies.

> **Caution**
> Do not edit the original source files. Rather, make copies of the source files, edit the copies, and run `plupdate` on the copies. Editing the original source files and running `plupdate` on them could corrupt your database.

To update a local file with another local file or to update a remote file with a local file, execute the `plupdate` utility from the command line as follows:

```
plupdate [-v] [-a] <dbname> <docID> <newrecfile>
```

To update a remote record with another remote record

1. Run `pldelete` on the doc ID of the record you are updating. For further information on `pldelete`, refer to "Deleting Records" on page 103.

2. Update the file with `plremote`. For further information on `plremote`, refer to "Adding Remote Files" on page 95.

The following table explains the command-line parameters for `plupdate`:

Table 8-6: Command-line Parameters: plupdate

Flag/Argument	Description	Explanation
`-v`	Verbose	Causes verbose output during processing.
`-a`	Append	Directs `plupdate` to append the new version of the replaced record to the container/source file that contains the previous version. If you use the `-a` option, PLWeb Turbo does not need the original `<newrecfile>` once indexing is complete; therefore you can discard or move it. If you do not use the `-a` option, the original file becomes part of the database and therefore, cannot be deleted or moved relative to the database. Do not use the `-a` option to update a record with a file having the same name as the original. If you wish to use this option, specify a new file name as the `<newrecfile>`.
`<dbname>`	Database Name	Specifies the name of the database to be updated.
`<docID>`	Document ID to be Replaced	Specifies the ID number of the record to be replaced.
`<newrecfile>`	New File	Specifies the path and name of the file to replace the record specified by `<docID>`. The contents of `<newrecfile>` must represent a single record and be compatible with the database's field definitions.

Example

In this example, the record with ID 99 in the `inventory` database will be replaced with the contents of the file `music2.txt`. Further, the contents of `music2.txt` will be appended to the original source file containing record 99, and PLWeb Turbo will no longer be concerned with the `music2.txt` file.

The original contents of record 99 will now be ignored, and the new contents will be read when searches are executed against the database.

```
plupdate -a inventory 99 c:\usr\db\music2.txt
```

8.13 Synchronizing a Database

The `plsync` utility simplifies the maintenance of dynamic, full-text databases. The `plsync` utility automatically updates databases based on the content of a directory tree. It monitors and notes any changes in that tree and it feeds the changes it detects to PLWeb Turbo. PLWeb Turbo then reflects those changes in the affected databases.

PLSync operates in two phases. In phase I, it determines which files need to be removed, added, and updated, and (optionally) generates a report. In phase II, it actually updates the database. Table 8-7 on page 108 denotes to which phase the configuration options apply. Some options may apply to only one phase.

PLSync can write the options for the specified database to a configuration file (if you specify the `-w` option) that is stored with the database. This allows the administrator to rerun `plsync` without having to respecify many of the options. Not all options are written to the configuration file, however. Only those options that are noted as “persistent” in Table 8-7 are written to the configuration file and only when you specify the `-w` option.

Execute the `plsync` utility from the command line as follows:

```
plsync [-v] [-tf|-tl|-tu|-tn] [-n] [-r] [-p <srcroot>] [-d] [-w] [-e <exclude-
file>] [-f <numfiles>] <dbname> ["<pat> {<pat>}"]
```

The following table explains the command-line parameters for `plsync`, and the following list explains the table’s columns:

- **Flag/Variable** Lists the flags and variables you may pass to `plsync`.
- **Description** Describes the flag/variable.
- **Phase Affected** Refers to whether a parameter is used to *determine* which files need to be removed, added, and updated (Phase I) or whether it actually updates the database based on the rules of Phase I (Phase II).
- **Persistent** Refers to whether a parameter is written to the database’s `plsync` configuration file.

- **Explanation** Explains the function of the flag/variable.

Table 8-7: Command-line Parameters: plsync

Flag/Variable	Description	Phase Affected	Persistent	Explanation
`-v`	Verbose	II	Yes	Causes verbose output while the database is being updated.
`-tf`	To File	I	Yes	Causes output to be written to two files: `_plrmfil.lst` and `_pladfil.lst`. These files contain the lists of files to be removed and added respectively, and can be used directly with `plrmfile` and `pladd`, using their `-f` options. This option (`-tf`) is the default when none of the target output options are supplied.
`-tl`	To List	I	Yes	Causes output to be written to standard out as a list of files that need to be added, removed, or updated. Files to be removed are prepended by a `-` sign, files to be added by a `+` sign, and files to be updated with a `*` sign. The main purpose of this output format is to allow Perl scripts to read the list from standard in. This option would normally be used together with the `-n` option, because the actual addition of files also causes output to be written to standard out.
`-tu`	To User	I	Yes	Causes output to be written to standard out with headers for files to be removed, added, and updated. This list is not suitable for machine interpretation.
`-tn`	To None	I	Yes	Causes no output to be generated concerning the list of files to be removed, added, and updated in the generation phase.
`-n`	No Update	II	Yes	Causes PLWeb Turbo to go through the process of creating the file lists to be removed, added, and updated, without actually updating the database.
`-r`	Recursive	I & II	Yes	Causes recursive processing of the root path.
`-p <srcroot>`	Path to Source Root	I & II	Yes	Specifies the root of the directory structure to process. If this is not supplied, `plsync` will find the root itself from the list of files already present in the database. A space must be present between `-p` and the path.
`-d`	Do Not Read Configuration File	I & II	No	Tells `plsync` *not* to read the configuration file for the specified database. If this option is not specified, `plsync` will attempt to read its configuration from this file. When it reads the configuration file, any command-line options that conflict with the settings in the configuration file will prevail.
`-w`	Write Configuration File	I & II	No	Specifies that the persistent options should be written to the configuration file for the specified database. If a configuration file is already present, it will read this file (unless the `-d` option was used), then process the command-line parameters. The resulting configuration settings will be written back to the configuration file.

Table 8-7: Command-line Parameters: plsync

Flag/Variable	Description	Phase Affected	Persistent	Explanation
`-e <excludefile>`	File to Exclude	I & II	Yes	Specifies the name of a file containing a list of files or directories to be excluded. Specifying a directory in this list when running `plsync` with the `-e` option will cause PLWeb Turbo to prune the directory tree at that point; subdirectories below the specified directory will not be processed. The format of the `<excludefile>` is one file or directory per line.
`-f <numfiles>`	File Count	I & II	Yes	Specifies the size of the hash table. Internally, `plsync` uses hash tables to do its list processing. The size of the hash tables (containing the file lists) is set to 2048. If the number of files greatly exceeds this number, processing may slow down. In order to optimize itself for a larger number of files, `plsync` will look for the `-f` flag. If it finds this, it will set the size of the hash tables to this value instead. This number does not need to be exact. However, it is recommended that, if you expect to process a number of files exceeding 10 times the default size (> 20000 files), you specify the `-f` option with a value between the number of files divided by 10 and the number of files.
`<dbname>`	Database Name	I & II	No	Specifies the name of the database to process. You must specify this parameter for `plsync` to find the correct configuration file.

Examples

```
plsync Help
plsync -r Help
plsync -tu Help
plsync -tu -p e:\plweb\plweb\databases\help\htmlsrc Help
```

8.14 Merging Databases

The need to merge a database often arises as a result of having a large quantity of data to index in a short period of time. The `plmerge` utility gives you an easy solution to solving this dilemma. You simply use two machines to index your data in parallel and then use `plmerge` to combine the resulting databases into one, previously existing (i.e., created prior to running `plmerge`) target database.

Indexing data in parallel and merging it into one database is faster than indexing your data in serial. The merge will take approximately 30% of the time it took to index the source data.[1] For example, assume it took four hours to serially index all of your data. If you had had two indexing jobs going (each indexing half of your data), it would have taken two hours to index the data into two databases and then approximately 36 minutes to merge one database into the other. Therefore, your total indexing time would have been two hours and 36 minutes as opposed to four hours.

1. Source data, in this instance, is the database that is being merged into the target database.

Note

For best performance, merge the smaller database into the larger database.

Execute the `plmerge` utility from the command line as follows:

```
plmerge [-v] [-t] <targetDatabase> <dbname> {<dbname>}
```

The following table explains the command-line parameters for `plmerge`:

Table 8-8: Command-line Parameters: plmerge

Flag/Argument	Description	Explanation
`-v`	Verbose	Causes verbose output during processing.
`-t <targetDatabase>`	Target Database	Specifies the target database into which all databases specified by `<dbname>` should be merged.
`<dbname>`	Database Name	Specifies the name of the database to be merged into the target database. Each database listed should be separated by a space or tab.

Example

The following example would merge the `HotelDB`, `MotelDB`, `BedBreakfastDB`, and `InnDB` databases into the target `AccomodationsDB` database. The target database, `AccomodationsDB`, must exist prior to running `plmerge`. After the databases are merged, the original databases remain unchanged.

```
plmerge -t AccomodationsDB  HotelDB  MotelDB  BedBreakfastDB  InnDB
```

Tip

If you wish to keep your data in date order, merge your most recent data into your older data (i.e., the older data will be the target database).

8.15 Reorganizing a Database Index

A PLWeb Turbo database index is self-optimizing; however, if desired, you can remove any unused space from a database's index by reorganizing its contents with the `plreorg` utility. You should reorganize a database only after it has undergone extensive updating and only if you are low on disk space; the process is unnecessary for a database that has had little updating.

If you have added partition statements to an existing database's definition file, you can, instead of completely reindexing the database, run `plreorg` to divide its index file into the specified partitions.

Before running the `plreorg` utility, you must copy the database's definition file (`.def`) to the new database's `<reorgname>.def` file in the same directory. If you wish to keep the original name for the reorganized database, you must rename `<reorgname>.pls`, `.adf`, `.acc`, `.syn`, and `.def` files *after* the reorganization is complete.

Execute the `plreorg` utility from the command line as follows:

```
plreorg [-v] [-{c|u}] <dbname> <reorgname>
```

The following table explains the command-line parameters for `plreorg`:

Table 8-9: Command-line Parameters: plreorg

Flag/ Argument	Description	Explanation
`-v`	Verbose	Causes verbose output during processing.
`-c`	Compact	Directs `plreorg` to compact the database to compacted form, in which it occupies the smallest possible amount of disk space. This option cannot be used in conjunction with the `-u` option.
`-u`	Uncompact	Directs `plreorg` to uncompact a previously compacted database prior to updating it. By specifying this option, you can avoid the prohibitive disk space overhead that is required when updating a database that is in compacted form. You should use this option only with a database that has been compacted; it cannot be used in conjunction with the `-c` option.
`<dbname>`	Database Name	Specifies the name of the database to be reorganized.
`<reorgname>`	Name of the New, Reorganized Database	Specifies the new path and name under which the reorganized database will be stored.

If you do not use the `-c` and `-u` options, the database index will be neither compacted nor uncompacted. It will be copied as it exists in its current state.

Updating a compacted database is extremely wasteful of disk space, as compared to doing so to an uncompacted one. Before updating a compacted database, you should first uncompact it by using `plreorg` with the `-u` option.

Example

In this example, the `fedlaw` database will be reorganized into compact form under the name of `lawopt01`.

```
plreorg -c fedlaw lawopt01
```

8.16 Deleting Databases

The `plrmdb` utility enables you to delete a database. For the database you specify, `plrmdb` will remove its reference from the `plweb.conf` file and will delete its `.pls` and other auxiliary files (e.g., `.acc` and `.adf`).

Execute the `plrmdb` utility from the command line as follows.

```
plrmdb [-v] [-d] [-f] <dbname>
```

The following table explains the command-line parameters for `plrmdb`:

Table 8-10: Command-line Parameters: plrmdb

Flag/ Argument	Description	Explanation
`-v`	Verbose	Causes verbose output during processing.
`-d`	Delete `.def` File	Directs `plrmdb` to delete the database's `.def` file as well. If you want to preserve the database's definitions for future use, do not specify this option.
`-f`	Force	Directs `plrmdb` to delete the database without prompting for confirmation.
`<dbname>`	Database Name	Specifies the name of the database to be deleted.

After deleting a database, you *must* restart the PLWeb daemon. If you do not immediately restart the daemon after deleting a database, PLWeb will not be able to successfully regenerate any children between the time a database is deleted, and the time the daemon is restarted.

Restart the daemon as follows:

1. From the START button, select Control Panel.

2. In the Control Panel window, double click the SERVICES icon.

3. In the Services window, select the `plwebd` service and click STOP and then, click START.

   After the server opens the local databases (you will get notification messages of this event), PLWeb Turbo will respond to user requests.

In addition, after deleting a database, you may want to

1. Update your templates, if the database names are hardcoded.

2. Update the `Databases` settings in the `view.conf` file.

Example

```
plrmdb -v -d Travel
```

8.17 Moving a Database

You can move a database by moving its files to a new location. All database files (other than the source files) can be found in the database directory (`<dbdir>`) (unless you partitioned the database). In addition to moving the database files, you must also move the source files so that they maintain the same relative location to the `<dbdir>` as before.

- A database is most portable when its index, definition, and source files all reside in the same directory or when the source files are stored in a subdirectory of the `<dbdir>`.

- If you know that you may move a database, you should, before indexing, position all source files in a central location or in locations that you can easily manage. If you move a database index to a new location, its source files must stay in the same relative position to it as when they were indexed.

- If, when adding or updating records, you specify absolute paths to the source files, you must recreate the database after moving it.

- If you move or rename a database's index file (`.pls`) and/or definition file (`.def`), you should update the database list by editing the appropriate line in `plweb.conf`.

- Determine whether the relocation affects any directories mapped in your relative links mapping file; update `baseurl.map` if necessary.

- Update your templates if the database was hardcoded in the templates.

9. Using Relative Links in HTML Files

HTML files that contain relative links present a special case for database administration with PLWeb Turbo, such that you will need to make some changes to your configuration and/or templates in order for PLWeb Turbo to resolve the relative links when your users retrieve these files. The simplest solution is to store all of your HTML source files in a subdirectory of the <drive>:\<installDir>\plweb\public-dbs directory. PLWeb Turbo is preconfigured to deal with relative links in source files that are stored in the public-dbs directory or any of its subdirectories.

If it is not possible to store the HTML source files in a subdirectory of <drive>:\<installDir>\plweb\public-dbs (for example, because there is not enough disk space available on the disk partition containing that directory), you can choose to store your source files in any other location. If you choose this option, in order for PLWeb Turbo to resolve the relative links, you must

1. Add an entry to the <drive>:\<installDir>\plweb\etc\baseurl.map file for the directory (or parent directory) in which the source files are stored (refer to page 118 for details).

2. Create a corresponding document alias in your HTTP daemon configuration file so your users can access the documents via hypertext link (refer to page 119 and your HTTP daemon documentation for more details).

In its default (shipping) configuration, PLWeb Turbo will resolve relative links at retrieval time by converting them into absolute links. There is a small amount of overhead associated with this process (specifically, when displaying a document) because PLWeb Turbo must first scan the source documents for relative links and then modify those links before the documents are sent to the browser.

You can gain a slight performance improvement by altering the method by which PLWeb Turbo resolves these links, but only if you know that none of the source documents contain the HTML <BASE HREF=...> tag. If no such formatting is present in the source files, you can turn off PLWeb Turbo's link parsing, and tell PLWeb Turbo to use an alternate method of link resolution, in which it adds a <BASE HREF=...> tag to each document at retrieval time. In order to achieve this, you must

1. Modify the <drive>:\<installDir>\plweb\views\<viewname>\view.conf file for the view that allows access to the database in question by setting the ModifyRelativeLinks entry to FALSE (refer to “Configuring the view.conf File” on page 83 for more details).

2. Modify the HTML document template (<drive>:\<installDir>\plweb\views\<viewname>\templates\doc_html.tmpl in the default configuration) to include the BASEURL macro within the <HEAD>...</HEAD> section (refer to the BASEURL macro on page 192 for more details).

9.1 Chapter Checklist

The following table provides a quick look at the options PLWeb Turbo provides for resolving relative links, based on whether you want to (1) support reference of BASE HREFs in your source files and (2) store HTML files containing relative links in the <drive>:\<installDir>\plweb\public-dbs directory.

You will find the following columns in the table. Consider the first two columns as input; they are conditions you can control. Consider the remaining columns as output; they are dependent on your input from the first two columns.

"Support BASE HREFs in Source Documents?" Do your source files already contain <BASE HREF=. . .> markup? Keep in mind that, if your documents include BASE HREF markup, your templates need to include absolute links.

"Store Source Files in public-dbs Directory?" Do you want to store all of your HTML source files in a subdirectory of the <drive>:\<installDir>\plweb\public-dbs directory? PLWeb Turbo comes preconfigured to deal with relative links in source files that are stored in the public-dbs directory or any of its subdirectories. If you store your HTML documents in a location outside of this directory, you will need to make some configuration changes.

"Use the BASEURL Macro" Do you want PLWeb Turbo to add an HTML BASE HREF tag to source documents? The BASEURL macro resolves paths on a file-by-file basis and is the most performance-friendly option of resolving relative links. For more information, refer to page 120.

"Set the ModifyRelativeLinks Setting in view.conf to" Directs PLWeb Turbo to turn on the modification of relative links. Setting this value to TRUE will cause PLWeb Turbo to change all relative links to absolute links. For more information, refer to page 121.

"Add a Mapping to the baseurl.map File" Tells PLWeb how to create a URL to your source files. You must modify this file if you do not store your HTML source documents in a subdirectory of the public-dbs directory. For more information, refer to page 118.

"Performance" Indicates the level of performance you will observe during indexing, as it relates to link parsing. Performance is slightly better when you use the BASEURL macro.

Table 9-1: Resolving Relative Links: Checklist

Input		**Output**			
Do You Want to. . .		**You will need to. . .**			**Keep in Mind that. . .**
Support BASE HREFS in Source Files?	**Store Source Files in public-dbs Directory?**	**Use the BASEURL Macro**	**Set the ModifyRelativeLinks variable in view.conf to**	**Add a mapping to baseurl.map[a]**	**Performance will be**
Yes	Yes	No	True	No	Normal
Yes	No	No	True	Yes	Normal
No	Yes	Yes	False	No	Improved
No	No	Yes	False	Yes	Improved

a. When you add a mapping to the baseurl.map file, you must also add one to your daemon resource configuration file. The baseurl.map file and the document aliases in the daemon are basically mirror images of each other. The difference is that the baseurl.map file resolves paths into base URLs, and the aliases resolve URLs into paths. For further information, refer to "Adding a Document Alias to the Daemon Resource Configuration File" on page 119.

9.2 Configuring the baseurl.map File

Located in <drive>:\<installDir>\plweb\etc directory, the baseurl.map mapping file contains rules that allow PLWeb Turbo to resolve paths to your source files into URLs. You must modify this file if you do not store your HTML source files in a subdirectory of the public-dbs directory.

Caution

Do not include symbolic links in the baseurl.map. They will not resolve correctly.

The baseurl.map file may contain rules and comment lines. Comment lines begin with a pound sign (#). Rule lines have the following syntax:

Format

```
<dir_path> <URL>
```

The following table explains the format you must use when defining the baseurl.map file:

Table 9-2: File Format: baseurl.map File

Argument	Description	Explanation
<dir_path>	Path to HTML File	Specifies an absolute directory path of the HTML document (retrieved by PLWeb Turbo) that contains relative links to be mapped to a URL.
<URL>	URL to Prepend to Relative Links	Specifies the mapped URL prefix (i.e., without the file name) that corresponds to the document directory. <URL> must be separated from <dir_path> with either a space or tab. A trailing / in the URL is required so that the complete URL will be formed properly. In addition, when dot notation (i.e., ..\<dir>\<file>) appears in a relative link, PLWeb Turbo will resolve the baseurl.map mapping and then climb the directory tree.

The matching strategy for the rules is first-fit; therefore, the order of the rules in the table is significant. In general, more specific rules should be listed before more general rules. A rule matching a parent directory is automatically inherited by a subdirectory. If no matching rule is found for a document, relative links in that document will not be resolved.

Example

```
# Sample mapping rules
c:\home\julie\dbs\news    http://www.abc.com/dbs/news/
c:\home\julie\dbs http://www.abc.com/julie/
```

Given these rules, the following mappings would take place:

Table 9-3: Example baseurl.map Mappings

The document PLWeb Turbo retrieves resides in	The retrieved document contains a link to	Using the baseurl.map file, the link is resolved to
c:\home\julie\dbs\news	budget.html	http://www.abc.com/julie/news/budget.html

Table 9-3: Example baseurl.map Mappings

c:\home\julie\dbs\	bermuda.html	http://www.abc.com/julie/bermuda.html
c:\home\julie\dbs\	..\logo.gif	http://www.abc.com/logo.gif

✉✉ Note ✉✉

Consider the order of the rules in the above example (a specific rule followed by a more general rule). If the rules were reversed, the link to budget.html (in the table) would resolve to http://www.abc.com/julie/news/budget.html, which would likely be incorrect.

As a further illustration using our example baseurl.map file, if an HTML document residing in c:\home\julie\dbs contains the following relative image link:

```
<IMG SRC="icons/big_abc_logo.gif">
```

PLWeb Turbo resolves the link to the following:

```
<IMG SRC="http://www.abc.com/julie/icons/big_abc_logo.gif">
```

9.2.1 Adding a Document Alias to the Daemon Resource Configuration File

Adding an entry to the baseurl.map file allows PLWeb Turbo to construct a URL referring to your source files. This does not guarantee that the HTTP daemon can actually access that URL. In order for the HTTP daemon to retrieve any files on your file system, it must be able to map the URL that is requested to a file to be retrieved. It accomplishes this through a set of document aliases in the HTTP daemon configuration file. The format required to set up a document alias in this file is different from daemon to daemon.

For example, when a user requests http://node:port/plweb/index.html, the HTTP daemon must know that this corresponds to <drive>:\<installDir>\plweb\htdocs.index.html on your file system. To let the daemon know about this relationship, you must add a document alias to your daemon configuration file (refer to your PLWeb Turbo installation notes for instructions). The mapping it adds is

```
plweb       <drive>:\<installDir>\plweb\htdocs\
```

(The first part of the URL, http://node:port, is defined elsewhere in the configuration files.)

The PLWeb Turbo installation comes preconfigured with the following single entry in the baseurl.map file, mapping source files in the <drive>:\<installDir>\plweb\public-dbs directory tree to URLs of the form http://node:port/plweb-dbs/:

```
<drive>:\<installDir>\plweb\public-dbs\    /plweb-dbs/
```

This entry tells PLWeb Turbo that any files in the <drive>:\<installDir>\plweb\public-dbs directory tree can be accessed by an URL of the form http://node:port/plweb-dbs/... (i.e., it

allows PLWeb Turbo to generate URLs of the form `http://node:port/plweb-dbs/...` for files in the <drive>:\<installDir>\plweb\public-dbs directory or its subdirectories).

When such a URL is generated and the browser displays the URL as a link to the user, the user can then click on the link. In order for the HTTP daemon to return the correct document, you must add the matching mapping to the HTTP daemon configuration file (refer to your PLWeb Turbo installation notes for instructions):

```
/plweb-dbs/        <drive>:\<installDir>\plweb\public-dbs\
```

If you add your own mappings to the `baseurl.map` file, you must add a matching (and opposite) entry into your HTTP daemon configuration file. For example, if you add the following mapping to `baseurl.map`

```
c:\home\user\mydb\sources\   /plweb-dbs/src/
```

your HTTP daemon configuration file should contain the following alias:

```
c:/plweb-dbs/src/     \home\user\mydb\sources\
```

In general, if you add

```
<directory>    <url>
```

to your `baseurl.map` file, you should add a document alias of the form

```
<url>      <directory>
```

to the HTTP daemon configuration file. Consult your HTTP daemon for information on the format required for the document alias.

✉✉ **Note** ✉✉

If you have database source files in multiple subdirectories of a single parent directory, and you want to make all of these accessible, you only have to add a single mapping for the parent directory to the `baseurl.map` file and the HTTP daemon configuration file

---

9.3 Using the BASEURL Macro

The BASEURL macro (page 192) resolve paths by adding the following markup to each file:

```
<BASE HREF="<URL>">
```

where

- <URL> is the URL used for resolving relative links. The mapping is performed by using the list of mappings in the `baseurl.map` file.

An alternative method for handling relative links in HTML documents is to use the ModifyRelativeLinks setting in the view.conf file, as discussed below. In that case, the BASEURL macro is no longer needed.

Example

The following example shows how you would use the BASEURL macro in your document screen template. The HEAD macro is included to retrieve the <HEAD>...</HEAD> section of the HTML document.

```
<HTML>
<HEAD>
{$BASEURL}
{$HEAD}
</HEAD>
. . .
```

9.4 Using the ModifyRelativeLinks Setting

The ModifyRelativeLinks setting in the view.conf file directs PLWeb Turbo to turn on the modification of relative links. Relative links that are present in HTML documents retrieved from a database will not work without either setting the ModifyRelativeLinks setting to TRUE or using the BASEURL macro. If you are using the BASEURL macro set the ModifyRelativeLinks setting to FALSE. For further information on the ModifyRelativeLinks setting, refer to “Configuring the view.conf File” on page 83.

10. Configuring the PLWeb Daemon

When the PLWeb Turbo daemon is started (`start-daemon`), it creates a parent process, which, in turn, creates multiple child processes. The parent not only creates child processes, it monitors their activity level and kills them when they are inactive for a specified amount of time (which is determined in the `server.conf` file). The parent does not service requests.

Child processes maintain communication with both the parent process and the user requests coming in via the PLWeb Turbo daemon's port. They tell the parent process how busy they have been and they service requests (e.g., pass a query or a request for a document) from the user.

You can control the manner in which parent and children behave by editing the default settings in the `server.conf` file, which is located in `<drive>:\<installDir>\plweb\etc`.

✉✉ **Note** ✉✉

The PLWeb Turbo daemon logs errors to `<drive>:\<installDir>\plweb\logs\error.log`.

10.1 Configuring the server.conf File

The `server.conf` file is a simple collection of key value pairs, one pair per line. Comments are allowed by using the # as the first symbol on the line. Keys are *not* case sensitive.

Table 10-1: Keywords and Values: server.conf

Keyword	Description	Key and Value Example
`Host`	Specifies the host on which the daemon runs.	`Host=www.abc.com`
`ServerPort`	Specifies the port used by the PLWeb Turbo daemon to communicate between parent and child processes.	`ServerPort=8098`
`HttpPort`	Specifies the port used by the HTTP daemon to communicate with the client (browser).	`HttpPort=80`
`CgiPath`	Specifies the portion of the URL after the node and port section pointing to the `<drive>:\<installDir>\plweb\cgi-bin` directory.	`CgiPath=plweb-cgi`
`Sessions`	Specifies the number of PLWeb Turbo sessions to create at startup. Every request PLWeb Turbo processes will need to access a session. On Windows NT, each request becomes a new thread. Because multiple threads cannont access the same session at the same time, if you are getting a large number of requests and have a low number of sessions, you will need to increase this setting so requests do not have to wait for free CPL sessions. However, if you set the number of sessions high and do not have a heavy load, you will consume resources unnecessarily	`Sessions = 5`

Sample server.conf File

```
# Name of host machine running the server.
Host=www.abc.com

# Port number the server will use internally.
ServerPort=8098

# Port number used by the HTTP daemon
HttpPort=80

# CGI path
CgiPath=plweb-cgi

# Number of sessions to open (each session corresponds to a thread.)
Sessions = 5

# The following settings do not apply to Windows NT.

# Allow no more than 16 children.
MaxProcesses=16

# Maintain at least 4 children.
MinProcesses=4

# Start/stop a single process at a time based on server activity.
ProcessIncrement=1

# Allow children to live 30 minutes.
ProcessLifetime=30

# Add new children if they are more than 90% busy.
MaxActivityThreshold=90

# Delete children if they are less than 10% busy.
MinActivityThreshold=10

# Check children every 90 seconds.
ActivityCheckInterval=90

# If a child doesn’t respond for more than one activity check interval
# in a row, delete it.
UnresponsiveLimit=1
```


11. Customizing Templates

The look, feel, and functionality of PLWeb Turbo is controlled by customizable HTML forms called interface *templates*. Any changes to the default shipping interface (e.g., adding your logo, offering special search options to your users, changing the way search results are presented to users, etc.) must be made to the templates.

In addition to containing HTML markup, templates also include *macros*. Similar to a function, a PLWeb Turbo macro can take arguments, get expanded/executed, and return an output string to the HTTP daemon. When it encounters a macro, PLWeb Turbo fills in the proper information, returns/writes that information, and continues to read the rest of the template. PLWeb Turbo does not associate specific macros with a template; therefore, any macro can be used in any template you develop, with the exception of those macros that are noted as template-specific in Appendix C.

Generally, every screen a user encounters is associated with an interface template (a few additional templates may be required as described below), and you can define as few or as many screens as is best for your application. Using the default templates as an example, the view selection screen is displayed to the user via the `views.tmpl` template. The search screen originates from the `search.tmpl` template.

Because PLWeb Turbo's templates allow a virtually unlimited range of options in creating your application, we recommend that you use the default shipping templates as your starting point. This chapter will provide a methodical, step-by-step process for enhancing those templates.

11.1 Designing Your Application

Consider following these basic steps when designing your application:

1. Determine the basic navigation of your application. What pages will your users encounter when using your application and in what order?

2. Create a flow diagram that outlines your application visually for you. This is a helpful reference when you have to implement your application.

3. Associate a template with every screen in your flow diagram.

4. Determine what additional templates are necessary.

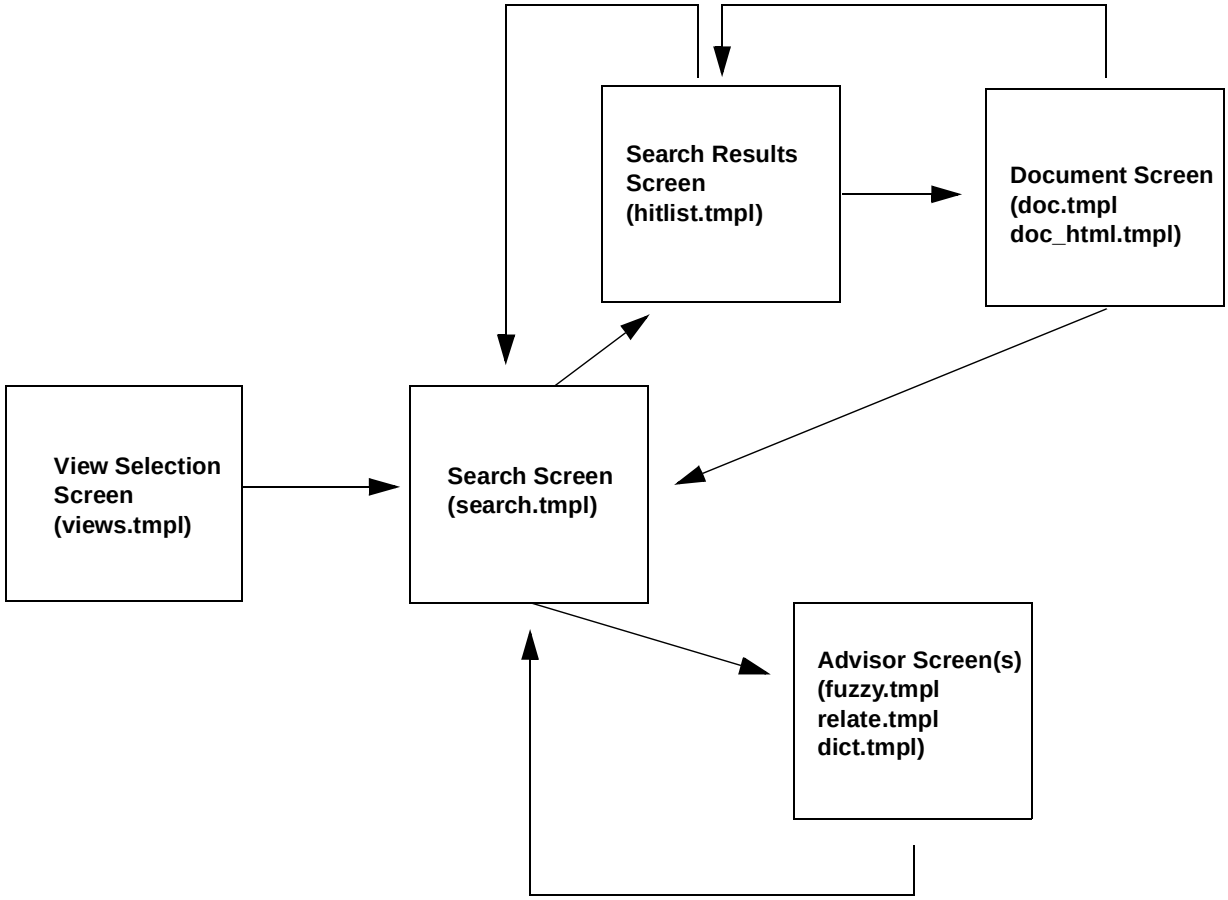
5. Proceed with implementing your application.

11.1.1 Determining Basic Navigation

The first step in designing your interface is deciding how users will navigate the different screens in your interface. You may decide to have all users start the search session through a universal search screen or you may provide several search screens for different types of users, such as basic and advanced. You might create specialized search screens for each unique type of information you provide, or offer a search box on the results list page. You may want users to go directly from the search results to the document screen, or you may want to display document summaries first. These are just a few examples of the options available to you.

11.1.2 Creating a Flow Diagram

Once you have considered the flow of your interface, sketch a diagram that represents the navigational choices you are providing. The following diagram shows the flow of the default PLWeb Turbo interface. In this example, each name represents a screen in the interface, and each arrow represents a link or button that allows the user to navigate to the connecting screen.



Note

Only screens with dynamic content are included in this diagram.

11.1.3 Associating Templates with Screens

After determining the flow of your interface, associate a template with every screen in the diagram in order to determine how many templates your interface requires. In the example above, we would create the following templates based on our flow diagram:

View Selection Screen	`views.tmpl`
Search Screen	`search.tmpl`
Search Results Screen	`hitlist.tmpl`
ASCII Document Screen	`doc.tmpl`
HTML Document Screen	`doc_html.tmpl`
Relate Screen	`relate.tmpl`
Dictionary Screen	`dict.tmpl`
Fuzzy Screen	`fuzzy.tmpl`
Error Screen (not shown in flow diagram)	`error.tmpl`

11.1.4 Determining the Need for Additional Templates

In certain cases additional interface templates may be necessary. When determining what additional templates are required use the following guidelines:

- You will need an additional template at each point in your flow diagram where the next template displayed is determined as a result of a user action.
- If you elect to program your site using HTML frames, you will need an additional template for each set of frames you create.
- You may wish to create additional templates to simplify complex templates or to isolate common code between all of your templates. The contents of these templates will be included in the original template during processing.

11.1.4.1 Results of User Action

An additional template is required whenever a user action determines the next template PLWeb Turbo must display. Because PLWeb Turbo reads and processes the templates *before* user action is taken, it cannot act on the results of the user action until *after* that action is complete. Therefore, there must be a “relay” template to evaluate the results of the user action and to tell PLWeb Turbo what to do next. For example, the search screen can go to either the advisor screens or the hitlist screen based on what the user selects. You cannot place the evaluating code (e.g., if the user selects the dictionary advisor, go to the `dict.tmpl` template) in the search screen template because at the time PLWeb Turbo reads the search screen template, the user action has not yet happened. You must create another template that follows the search screen template and

1. Asks PLWeb Turbo, what the user selected. A simple search? An advisor search?

2. Tells PLWeb Turbo what to do based on the result: if the user selected a search, display the hitlist template; if the user selected an advisor, display an advisor template.

The PLWeb Turbo sample templates include the `prehit.tmpl` template to handle this hitlist/advisor routing. The search screen targets `prehit.tmpl`, which in turn calls either the hitlist screen or one of the advisor screens.

Another example that can be derived from PLWeb Turbo's default application is as follows. Assume the user clicks on a hit in the hitlist. Because the documents in the hitlist may require either the ASCII or HTML document display template, another "relay" template is required. When PLWeb Turbo creates the hitlist, it does not know the document format of each hit. So, when the user clicks on a hitlist link the "relay" template will

1. Ask PLWeb Turbo the format of the document the user selected. ASCII? HTML?

2. Tell PLWeb Turbo what to do based on the result. If the document is ASCII, display the `doc.tmpl` template; if the document is HTML, display the `doc_html.tmpl` template.

The PLWeb Turbo sample templates include `predoc.tmpl` to handle this ASCII/HTML document display choice. The hitlist template targets `predoc.tmpl`, which in turn calls the appropriate document display template.

11.1.4.2 HTML Frames

Standard HTML frames are composed of an HTML page containing the frameset and two or more pages containing the contents of the individual frames. In order to use frames with PLWeb Turbo, you must create the frameset as a template rather than as an HTML document. This frameset will become the target of an operation in PLWeb Turbo, and one or more of the component frames in the frameset will be a URL interpreted by PLWeb Turbo. Therefore, if your interface includes HTML frames, an additional template is necessary to represent each HTML frameset. Refer to "More on Using HTML Frames" on page 150 for examples of implementing frames.

11.1.4.3 Complex Templates with Common Code

PLWeb Turbo allows you to organize template functionality in separate files that will be read into a single template when they are processed. This is optional and can be useful for example, when

- Your templates are complex and it would be easier to manage them and find errors if you could isolate sections to a discrete file. For example, you want to deal with the HTML aspects of the page separately from the PLWeb Turbo functionality in the page. You can create the HTML look and feel in one template and place the PLWeb Turbo macros in another template. The HTML "look-and-feel" template can then be scripted to include the template containing the macros when PLWeb Turbo reads it (the look-and-feel template) in.

- Your templates share common code that is difficult to maintain when it is repeated in many places. For example, rather than editing each of your templates to change the color or placement of your logo, you could create a separate template containing the color and logo definition and include that single template in all interface screens.

The PLWeb Turbo sample templates use this option extensively. All of the following templates were created to isolate functionality and common code in separate files. The templates were then included in their respective parent templates via the INCLUDE macro, which is discussed on page 209.

Logo, look-and-feel code	logo.tmpl
View Selection Screen implementation	views_impl.tmpl
Search Screen implementation	search_impl.tmpl
Search Results Screen implementation	hitlist_impl.tmpl
ASCII Document Screen implementation	doc_impl.tmpl
HTML Document Screen implementation	doc_html_impl.tmpl
Relate Screen implementation	relate_impl.tmpl
Dictionary Screen implementation	dict_impl.tmpl
Fuzzy Screen implementation	fuzzy_impl.tmpl
Error Screen implementation	error_impl.tmpl

For additional information, refer to the INCLUDE macro on page 209.

11.2 Implementing the Interface

Once you have determined the structure of your application, you are ready to implement your templates. You may approach this in one of two ways.

1. If some of your screens are functionally similar to the screens that ship with PLWeb Turbo (e.g., your search screen is followed by hitlist display and then document display) you may find it easiest to start with the default templates and modify them to suit your design.

2. If your application is fundamentally or substantially different from PLWeb Turbo, you can start from scratch, using PLWeb Turbo's default templates as a reference.

Regardless of which method you choose, it is important that you read the next sections to understand the structure of the default PLWeb Turbo templates.

11.2.1 Implementing Templates: The Basic Steps

The following section outlines the basic procedure for implementing templates.

1. Using standard HTML scripting (or HTML authoring tools) and a Web browser, create the look and feel of the pages users will come to know as your interface—refer to page 133.

2. Create the shell of all other required auxiliary templates (if needed), such as “relay” or frame templates and name them accordingly—refer to page 133.

3. Link the standard templates together by inserting target statements.

4. Complete the “relay” templates and frame templates.

5. Add the remaining PLWeb Turbo searching functionality to pages with established look and feel. We recommend using the `INCLUDE` macro (refer to page 209) to simplify complex pages—refer to page 135.

6. Test your pages with PLWeb Turbo.

11.2.2 Designing the Look and Feel

Because the templates consist of standard HTML code in addition to PLWeb Turbo macros for dynamic content, you are able to implement and test the look and feel of your screens independent from PLWeb Turbo functionality. Create HTML files that include your logo or your application's menus or toolbars, using dummy elements where you want PLWeb Turbo to insert dynamic content. As long as you do not add any macros to these pages at this time, you can test your screens directly with your browser or with the browser and the HTTP daemon.

11.2.3 Creating Auxiliary Templates

If your application requires auxiliary templates such as “relay” or frame templates, create blank files now and give them functional names that will make sense to you and others. For example, our document display routing template is called `predoc.tmpl`, because it precedes document display. You will go back and implement the code in these templates after you have completed the next few steps.

11.2.4 Linking Templates Together

In order for PLWeb Turbo to traverse the templates the way you intended, you must provide pointers that reference the correct target template. Without target statements in your templates, PLWeb Turbo will not know what template to display next and will display a blank screen.

✉✉ **Note** ✉✉

Do not forget that you are targeting the file that PLWeb Turbo needs to read next, not the file the user will see next. For example, if you require a “relay” template between the search screen and the hitlist/advisor screens, the search screen must target the “relay” template not the hitlist or advisor templates.

11.2.4.1 Defining a Target Template from a Form

If the current template displays a form, and you need to define where to go when the form is submitted, add the following markup to the current template, within the `<form>...</form>` section:

```
<input type=hidden name=TemplateName value=<TargetTemplate>>
```

where

- <TargetTemplate> is the name of the target template.

11.2.4.2 Defining a Target Template from a Link-Generating Macro

In order to define where PLWeb Turbo should go when the user clicks on a link generated by PLWeb Turbo, you must add a target="<TargetTemplate>" argument to the macro generating that link.

Example

```
{$DOCLINK: target=predoc.tmpl}
```

11.2.5 Completing the “Relay” and Frame Templates

11.2.5.1 “Relay” Templates

Previously (under “Creating Auxiliary Templates” on page 133), you created a shell template for each point in your application where a user action determines the next template PLWeb Turbo must display. Now, you will provide the logic that will tell PLWeb Turbo what template to display.

Include the following lines in the shell “relay” template you created to provide PLWeb Turbo with information to determine the target template.

```
{$IF: variable=<variable>, operator=<operator>, value=<value>}
    {$INCLUDE: file=<TemplateName1>}
{$ELSE}
    {$INCLUDE: file=<TemplateName2>}
{$ENDIF}
```

where

- <variable>, <operator>, and <value> are the conditions that must be satisfied for the first listed template to be invoked. Arguments for <variable>, <operator>, and <value> are predefined for the IF macro, as described on page 205.

- <TemplateName1> is the template PLWeb Turbo will call when the condition is satisfied.

- <TemplateName2> is the template PLWeb Turbo will call if the condition is not satisfied.

For further information on the IF, ENDIF, INCLUDE, and ELSE macros, refer to Appendix C.

Example

The following example would display either template A or template B, depending on the user's selection in the previous template.

```
{$IF: condition=querytype, operator=eq, value=simplesearch}
    {$INCLUDE: file=a.tmpl}
{$ELSE}
    {$INCLUDE: file=b.tmpl}
{$ENDIF}
```

✉✉ Note ✉✉

If you have more than two options in your “relay” template, use a nested `IF` statement.

Example

The following is the text of the `prehit.tmpl` template as it is shipped with PLWeb Turbo. It is the file PLWeb Turbo's search screen invokes to determine which template to display after the user has selected between a standard search and advisor search operators.

```
{$IF: variable=querytype, operator=eq, value=dictionary}
    {$INCLUDE: file=dict.tmpl}
{$ELSE}{$IF: variable=querytype, operator=eq, value=relate}
    {$INCLUDE: file=relate.tmpl}
{$ELSE}{$IF: variable=querytype, operator=eq, value=fuzzy}
    {$INCLUDE: file=fuzzy.tmpl}
{$ELSE}
    {$INCLUDE: file=hitlist.tmpl}
{$ENDIF}
{$ENDIF}
{$ENDIF}
```

11.2.5.2 Creating Frame Templates

If you are using HTML frames, you will need to complete the shell of the frame template you created (refer to “Creating Auxiliary Templates” on page 133) before implementing frames.

The PLWeb Turbo template you create for the frameset will be slightly different than an ordinary HTML frameset page. For frames that will simply display static content, like a menu, you can reference that static page as you would with standard HTML. For frames that will contain dynamic, PLWeb Turbo-generated content, your reference must be in the following format:

```
<a href="http://www.xyz.com/plweb-cgi/fastweb?view={$VIEWNAME}&TemplateName
"<TemplateName>">
```

where

- `<TemplateName>` is the name of the template to which you are referring.

For additional information on using HTML frames, refer to “More on Using HTML Frames” on page 150.

11.2.6 Adding PLWeb Turbo’s Searching Functionality

Once you have developed templates with the appropriate look and feel, you are ready to add dynamic content to the templates by using any of the macros described in Appendix C.

11.2.7 Simplifying Complex Templates

If the look and feel of your screens is complex, and the dynamic content is largely clustered together in one portion of each template, you may want to consider simplifying individual templates by using the INCLUDE macro, which is described on page 209. This macro allows you to include, by reference, one template in another. Internally, the templates are processed as if the included template were part of the parent template. The default PLWeb Turbo templates use this macro extensively.

Example
The following example shows the shipping PLWeb Turbo search screen template. The first INCLUDE (logo.tmpl) incorporates a "look-and-feel" template, the second INCLUDE (search_impl.tmpl) incorporates a template that contains PLWeb Turbo functionality.

```
<HTML>
<HEAD>
<TITLE>PLWeb Turbo Search</TITLE>
</HEAD>
<BODY BGCOLOR="#ffffff" BACKGROUND="/plweb/icons/subback.jpg" TEXT="#001572"
LINK="#008800" ALINK="#000000" VLINK="#bb4400">
<TABLE BORDER="0" CELLSPACING="0" CELLPADDING="0" WIDTH="600">
<TR>

{$INCLUDE: file=logo.tmpl}

<TD VALIGN="TOP">
<IMG SRC="/plweb/icons/search_word.gif" WIDTH="267" HEIGHT="48" BORDER="0"
ALT="Search" NATURALSIZEFLAG="0" ALIGN="BOTTOM"></TD>
<TD VALIGN="TOP">
<IMG SRC="/plweb/icons/fillline.gif" WIDTH="173" HEIGHT="48" BORDER="0" ALT="-
----" NATURALSIZEFLAG="0" ALIGN="BOTTOM"></TD>
<TD VALIGN=TOP ROWSPAN="2"><IMG SRC="/plweb/icons/search_icon.gif" HEIGHT=72
WIDTH=102></TD>
</TR>

{$INCLUDE: file=search_impl.tmpl}

<TR>
<TD WIDTH="17%"> </TD>
<TD WIDTH="17%"> </TD>
<TD WIDTH="17%"> </TD>
<TD WIDTH="17%"> </TD>
</TR>
</TABLE>
</BODY>
```

11.2.8 Testing Your Pages

At this point, you should have a working set of templates. To test your templates, start the daemon and access the newly created pages. Most likely, you will want to make modifications, view the results, make modifications again, etc. Remember that you need to restart the PLWeb Turbo daemon (not the

HTTP daemon) every time you make a change you want to see in action. PLWeb Turbo reads all templates at start-up to improve performance.

To restart the PLWeb Turbo daemon:

1. From the START button, select Control Panel.

2. In the Control Panel window, double click the SERVICES icon.

3. In the Services window, select the `plwebd` service and click STOP and then, click START.

   After the server opens the local databases (you will get notification messages of this event), PLWeb Turbo will respond to user requests.

✉✉ **Note** ✉✉

While you are customizing your templates, you may want to consider turning off preparsing of templates by adding the key and value `PreparseTemplates=False` to the `view.conf` file (refer to Table 7-2 on page 83). Turning off preparsing of templates enables you to modify and debug templates without having to restart the daemon for changes to those templates to take affect (i.e., any changes you make will be effective immediately). When you have finished customizing your templates, turn preparsing of templates back on by removing the `PreparseTemplates` key and value from the `view.conf` file.

11.3 Adding Functionality

The following section describes the functionality you can add to your interface with the PLWeb Turbo macros. It does not provide information or examples of HTML scripting beyond that which appears in the default templates in Appendix B. It is assumed that, through (1) the information we provide about PLWeb Turbo macros, (2) your knowledge of HTML, and (3) the use of the default templates as examples, you are able to create both a functional and aesthetically pleasing search application.

✉✉ **Note** ✉✉

The following tables group macros by their *likely* functionality. To conserve space, most macros have not been repeated in multiple tables. This does not mean, however, that the macros cannot cross tables and thereby, functionality.

Table 11-1: View Display and Database Display Functionality

Function to Implement	Macro to Use	Additional Information
Display the database description	`DBDESC`	page 194
Display all databases in this view.	`DBLIST` `DLIST_END`	page 194 page 195

Table 11-1: View Display and Database Display Functionality

Function to Implement	Macro to Use	Additional Information
Display a database.	DBNAME	page 195
Display the view description.	VIEWDESC	page 227
Display the view ID.	VIEWID	page 227
Display all views available to a user.	VIEWLIST VIEWLIST_END	page 227 page 228
Display a view name.	VIEWNAME	page 228

Table 11-2: Hitlist Display Functionality

Function to Implement	Macro to Use	Additional Information
Display the path and name of a document's container.	CONTAINER	page 193
Display a document's ID.	DOCID	page 198
Display a document's size.	DOCSIZE	page 199
Display the rank of the first-ranked document in the current section of the hitlist.	FIRSTHITRANK	page 201
Display a list of search results (hitlist).	HITLIST HITLIST_END	page 203 page 204
Display the number of documents found.	HITSFOUND	page 204
Display the number of documents returned.	HITSRETURNED	page 204
Display the rank of the last hit in the hitlist.	LASTHITRANK	page 210
Display the number of documents requested.	MAXITEMS	page 210
Display the number of documents on the next screen.	NEXTNUMHITS	page 211
Display a document's normalized relevance score.	NSCORE	page 212
Display the number of documents on the previous screen.	PREVNUMHITS	page 214
Display the query that generated the current results.	QUERY	page 215
Display a document's rank in the hitlist.	RANK	page 216
Display a document's raw relevance score.	SCORE	page 218
Display a graphical representation of a document's relevance score.	SCOREIMG	page 219
Display a document's summary.	SUMMARY	page 223
Display a document's title.	TITLE	page 224

Table 11-3: Document Display Functionality

Functionality to Implement	Macro to Use	Additional Information
Display a `base HREF` statement in HTML documents that contain relative links.	`BASEURL`	page 192
Display the contents of a document.	`DOC`	page 197
Display the `HEAD` portion of an HTML document.	`HEAD`	page 202
Display the query that generated the current document.	`QUERY`	page 215
Display highlighted word in a document.	`SDETAIL_HIT` `EOL`	page 221 page 200
Display a link to jump to the first highlighted word in a document.	`SDETAIL_LINK` `EOL`	page 221 page 200
Display the number of times the current word occurs in document.	`SDETAIL_OCCUR` `EOL`	page 221 page 200
Start a loop to iterate over the highlighted words in the document.	`SDETAILLIST` `SDETAILLIST_END`	page 220 page 220

Table 11-4: Advisor Display Functionality

Functionality to Implement	Macro to Use	Additional Information
Create a loop that repeats over a list of dictionary advisor terms.	`DICTLIST` `DICTLIST_END`	page 196 page 196
Display a dictionary term.	`DICTTERM`	page 196
Display a dictionary term ID.	`DICTTERMID`	page 197
Create a loop that repeats over a list of fuzzy advisor terms.	`FUZZYLIST` `FUZZYLIST_END`	page 201 page 201
Display a fuzzy term.	`FUZZYTERM`	page 201
Display a fuzzy term ID.	`FUZZYTERMID`	page 202
Display the number of times a dictionary term occurs in the selected database.	`OCCUR`	page 212
Create a loop that repeats over a list of relate advisor terms.	`RELATELIST` `RELATELIST_END`	page 217 page 217
Display a relate term.	`RELATETERM`	page 217
Display a relate term ID.	`RELATETERMID`	page 217

Table 11-5: Navigational Functionality

Functionality to Implement	Macro to Use	Additional Information
Link to a document.	DOCLINK EOL	page 198 page 200
Link to the next document in the hitlist.	NEXTHIT EOL	page 210 page 200
Link to the next section of the hitlist.	NEXTLIST EOL	page 211 page 200
Link to the previous document in the hitlist.	PREVHIT EOL	page 213 page 200
Link to the previous section of the hitlist.	PREVLIST EOL	page 214 page 200
Link to a list of documents similar to the current document.	QUERYBYEXAMPLE EOL	page 216 page 200
Link directly to a remote document (no highlighting).	RMTLINK EOL	page 218 page 200
Link directly to a source document, to be returned in its native format.	SRCLINK EOL	page 222 page 200
Link to the first hit word in a document.	TOFIRSTHIT EOL	page 225 page 200
Link to the list of search results (hitlist).	TOHITLIST EOL	page 226 page 200
Link to the search screen.	TOSEARCHFORM EOL	page 226 page 200

Table 11-6: Error Message Display

Functionality to Implement	Macro to Use	Additional Information
Display an error ID.	ERRORID	page 200
Display the text of an error message.	ERRORMSG	page 200

Table 11-7: Logging, Date, and Pricing Functionality

Functionality to Implement	Macro to Use	Additional Information
Display a custom log string.	CUSTOMLOGSTRING	page 194
Display the price of a document.	PRICE	page 214
Display the local time.	TIMESTAMP	page 223

Table 11-8: Other Functionality

Functionality to Implement	Macro to Use	Additional Information
Display the value of a CGI variable.	CGIVAR	page 192
Display the query being passed to CPL.	CPLQUERY	page 193
Terminate a hypertext link.	EOL	page 200

Create ELSE and IF control blocks.	ELSE
	IF
	ENDIF
		page 199
		page 205
		page 199

Preserve user settings.	HIDDENVALS	page 203
Include another template into the current template.	INCLUDE	page 209
Display an advertisement.	INSERTAD	page 209
Redirect output to a specified destination.	OUTPUT	page 213
Preserve database-selection settings.	SELECTEDBS	page 222

11.3.1 Creating the Query Rule

The query rule is an element in any template that enables the user to enter a query (e.g., the search screen). The query rule is an expression that gets expanded based on the query the user has entered in query text boxes.

The query rule can be thought of as a list of expressions, each describing a component of the query. A component may contain (and usually does contain) a variable of the form ($query). The simplest query rule would consist of a non-fielded component, while more complex query rules would consist of non-fielded, field-restricted, and range components.

✉✉ Note ✉✉

The query rule applies to both hardcoded and user-entered queries.

The following tables list the various components of the query rule. Components, functions, and instructions on combining them will be discussed in more detail in the following text.

Table 11-9: Query Rule Components

Component	Description	Example(s)
Simple, Non-fielded	Specifies that all query terms are run against all searchable fields. The most basic form of the query rule.	($query)
Field Restriction Component	Specifies that a component of the query rule is associated with a particular field.	(($query0)):TITLE or (($query)) AND (($query0)):<FIELD_NAME>

Table 11-9: Query Rule Components

Component	Description	Example(s)
Range Component	Specifies a numeric range search in which two components of the query rule determine the boundaries of valid search results.	(($query1).le.<FIELD_NAME>.le.($query2))

✉✉ Note ✉✉

Range expressions must always be surrounded by parentheses.

Table 11-10: Query Rule Functions

Type	Format	Example
Date	#date(variable)	(#date($query0).gt.<FIELD_NAME>.gt.#date($query1))
Price	#price(variable)	(#price($query0).le.PRICE.le.#price($query1))

11.3.1.1 Combining Query Rule Components and Functions

Regardless of your query rule's complexity, it must always conform to the following base format:

```
<INPUT TYPE=hidden NAME=query_rule VALUE="<value>">
```

where

- <value> is the query rule.

✉✉ Note ✉✉

1.You may only have one query rule per template. (A query rule commented out with HTML comments is considered a second query rule, regardless of the comments.)
2.The query rule is case insensitive.

Non-Fielded Component

The non-fielded component comprises the simplest form of the query rule and is always the first building block of the query rule when combining multiple components (refer to page 142).

If you had a single, non-fielded query input field in a template, you would create a query rule containing a single variable expression of the following form:

```
($query)
```

> **✉✉ Note ✉✉**
>
> The non-fielded component of the query rule must be specified as `($query)`. It cannot be any other word or in any other format.

The complete query rule would be

```
<input type=hidden name=query_rule value="($query)">
```

If the user entered *surfing in Hawaii* as his/her query, PLWeb Turbo would parse the query rule and replace the `($query)` variable with the query the user entered, creating a single query (in this case, *surfing in Hawaii*).

> **✉✉ Note ✉✉**
>
> In the absence of a query rule in a template, PLWeb Turbo automatically creates a query rule consisting of the non-fielded query rule (`($query)`). If a query rule is present, however, it will be used as the rule submitted to PLWeb Turbo.

Field-Restricted Component

The field-restricted component of the query rule takes the following format:

```
($query<N>)
```

where

- `<N>` must be a value between 0 and 63 (inclusive).

> **✉✉ Note ✉✉**
>
> In its most minimal format, the fielded component of the query rule must be specified as `($query<N>)`. It cannot be any other word or in any other format other than`($query<N>):<field_name>`, as shown below.

You can combine fielded components either with operators (refer to page 230) or by simply appending one to another (in which case PLWeb Turbo will implicitly insert the default operator defined in `view.conf` between components). In order to associate a specific component of the query rule with a particular field in your `.def` file, you would need to add an expression of the following form to the query rule:

```
($query<N>):<field_name>
```

where

- <N> must be a value between 0 and 63 (inclusive), used internally by PLWeb Turbo. If this is the first field-restricted component you have defined, <N> must be 0. If it is the second, <N> must be 1, etc.

- <field_name> is a field or list of fields to which the search should be restricted. If specified as a list, the items in the list must be comma separated.

For example, assume that you have an AUTHOR field in your database and query0 defined in one of your templates. The expression describing this section of the query would be similar to

```
(($query0)):AUTHOR
```

✉✉ **Note** ✉✉

Two sets of parentheses are required. The inner set is part of the variable to be expanded, whereas the outer set will be part of the query after expansion by PLWeb Turbo. This outer set is required to make sure that the field restriction—indicated by the colon in :AUTHOR—applies to all terms in query0.

A complete query rule combining non-fielded and field-restricted components might be

```
<input type=hidden name=query_rule value="(($query)) AND (($query0)):AUTHOR">
```

Given the above query rule, if the user input *The Bridges of Madison County* as a non-field-restricted query and *Robert James Waller* as a field-restricted query against the AUTHOR field, PLWeb Turbo would expand the query as follows:

```
(The Bridges of Madison County) AND (Robert James Waller):AUTHOR
```

Range Component

The range component is an optional part of the query rule, specifically available to support *range searching*. Range searching allows users to limit their searches to documents with field contents that fall within a certain range. For example, if a database contained newspaper articles, users might want to limit their searches to dates within a certain range.

The expression describing a range component may take one of the following forms:

```
<value> <comparator> <field_name>
<field_name> <comparator> <value>
<value> <comparator> <field_name> <comparator> <value>
```

where

- <value> is a numeric value.

- <field_name> is the name of a valid field in the databases definition file (.def).

- <comparator> is one of the following values:

`.lt.` (less than)
`.le.` (less than or equal)
`.gt.` (greater than)
`.ge.` (greater than or equal)

For example, assume you want to allow the user to input both a minimum and a maximum value for the date of the documents to include in a search. If the field that contains the document date is called DATE, the corresponding range component in the query rule would be

```
($query1).le.DATE.le.($query2)
```

✉✉ **Note** ✉✉

In the above example, only a single set of parentheses is required around the variables because they should expand to a single value.

An example of a complete query rule combining non-fielded and range components would be as follows:

```
<input type=hidden name=query_rule
value="(($query)) AND ($query1).le.DATE.le.($query2)">
```

Date Function

In order for PLWeb Turbo to successfully execute searches on dates, you must store dates in their own field (refer to page 54) in the following standard format:

```
<YYYY><MM><DD>
```

where

- <YYYY> is the four-digit year.
- <MM> is the two-digit month of the year.
- <DD> is the two-digit day of the month.

Because this is not a user friendly format, PLWeb Turbo provides a date conversion function to allow users to enter dates as they would naturally, while still enabling standardized range searches on dates in the database. PLWeb Turbo is able to convert a “natural” date to a standard format using the following function:

```
#date
```

The date function will attempt to translate a date entered in any common format into the standard format. For example, PLWeb Turbo would correctly interpret the following dates:

```
11/26/93
11-26-93
```



```
Nov. 26, 1993
26 nov 93
```

If a European user entered `26/11/93`, that date would also be expanded correctly; however, in the event of interpretive conflict (e.g., `11/10/93`), PLWeb Turbo would assume U. S. date format.

Incorporated into the query rule, the date function would take on the following format:

```
#date(<variable>)
```

where

- `<variable>` represents either a non-fielded or field-restricted query rule component (refer to page 142): `($query)` or `($query<N>)`.

Example
If you wanted to enable the user to input the date in a natural format, the query rule would change to

```
<input type=hidden name=query_rule
value="(($query)) AND (#date($query1).le.DATE.le.#date($query2))">
```

If the user were to enter `26 November 1993`, PLWeb Turbo would first expand `($query1)` to `26 November 1993` and then convert `#date(26 November 1993)` to `19931126`.

✉✉ **Note** ✉✉

If you anticipate that users will be entering dates in the format of `<YYYY><MM><DD>`, it is recommended that you not use the date function. If you use the date function and users enter dates as they are stored in your source files, PLWeb Turbo will not process them correctly.

✉✉ **Note** ✉✉

PLWeb Turbo correctly handles the year 2000 and following years correctly. For example, the date `12/31/00` would be correctly interpreted as `December 31, 2000` and convert it to `20001231`.

Price Function

In order for PLWeb Turbo to successfully execute searches on prices, you must store prices in their own field (refer to page 55) in integers representing the smallest currency unit available for your locale (e.g., cents or pennies). For example, a document costing $1.50 would be represented by `150` in the price field.

Because this is not a user friendly format, PLWeb Turbo provides a price conversion function to allow users to enter prices as they would naturally, while still enabling standardized range searches on prices in the database. PLWeb Turbo is able to convert a “natural” price to a standard format using the following function:

#price

Incorporated into the query rule, the price function would take on the following format:

#price(<variable>)

where

- <variable> represents either a non-fielded or field-restricted query rule component: ($query) or ($query<N>).

✉✉ **Note** ✉✉

If you anticipate that users will be entering prices in the same format that you have used for prices in your source files, it is recommended that you not use the price function. If you use the price function and users enter prices as they are stored in your source files, PLWeb Turbo will not process them correctly.

Example
If you wanted to enable the user to input the price in a natural format, the query rule would change to

```
<input type=hidden name=query_rule
value="(($query)) AND (#price($query1).le.PRICE.le.#price($query2))">
```

If the user were to enter $1.50, PLWeb Turbo would first expand ($query1) to $1.50 and then convert #price($1.50) to 150.

11.3.1.2 Combining Components and Functions

In case some components are optional and cannot be expanded to a non-empty value, those components will be removed automatically during query rule parsing. The following section illustrates, through a scenario of potential user activity, the manner in which PLWeb Turbo would parse the query rule given the user's input.

Example
($query)) AND (#date($query0).le.DATE.le.#date($query1))

Suppose the user did not enter anything corresponding to the second range component, the above query rule would be equivalent to the following query rule:

(($query)) AND (#date($query0).le.DATE)

Suppose the user did not enter anything corresponding to the first range component, the above query rule would be equivalent to the following query rule:

(($query)) AND (DATE.le.#date($query1))

Suppose the user did not enter anything corresponding to the first and second range components, the above query rule would be equivalent to the following query rule:


```
(($query))
```

Suppose the user did not enter anything corresponding to the non-fielded component, the above query rule would be equivalent to the following query rule:

```
#date($query0).le.DATE.le.#date($query1))
```

11.3.2 Setting Hitlist Sorting

You can pre-define a set of template macro options to sort the hitlist by relevance, by field content, or not at all. End users can then select from among these pre-defined options.

Hitlist sorting is determined by an HTML setting named `sorting` (to see an example, refer to the `search_impl.tmpl`, on page 172). The value of this setting can be one of the following:

```
NONE
BYRELEVANCE
BYFIELD:[+|-]<FIELD>
```

- If you set `sorting` to `NONE`, for a single local database, the hitlist will be sorted in reverse document ID order.
- If you set `sorting` to `BYRELEVANCE`, your hitlist will be sorted by relevance.
- If you set `sorting` to `BYFIELD`, a definition of how sorting should be achieved must be included in the value of the setting. The definition is a list of field specifications, separated by commas. The format for each field specification is as follows:

```
[+|-]<FIELD>
```

where

- A `+` sign denotes ascending order and a `-` sign denotes descending order.
- `<FIELD>` is the name of a field defined in the database's database definition file.

An example of an implementation allowing the user to select between no sorting, relevance sorting, and field sorting is shown below:

```
<select name=sorting>
<option value="NONE"> No sorting
<option value="BYRELEVANCE"> By relevance
<option value="BYFIELD:+AUTHOR,-DATE"> By Field
</select>
```

✉✉ Note ✉✉

Any desired HTML control can be used to select the sorting method (e.g., radio buttons, drop-down controls, check boxes, etc.).

11.4 Adding Copyright Information to Your Templates

Adding copyright information to your PLWeb Turbo screen templates is a simple exercise in standard HTML markup. Depending on the nature of your PLWeb Turbo screens, you may need to place the same copyright information on one or all or your pages, or you may need to customize the copyright information. Regardless of the content, we recommend that you position your copyright information in the same location on each page, in order that it become a familiar yet unintrusive element of your display.

Example
The following is an example for adding copyright information to your screen templates. It assumes the presence of an `<HTML>` marker at the top of the page, above the macros and fields you used for configuring template functionality.

```
<HR>
Copyright 1998 America Online Inc. All Rights Reserved.<P>
PLWeb Turbo is a registered trademark of Personal Library Software Inc.<P>
</HTML>
```

11.5 Configuring the Online Help

Your PLWeb Turbo distribution is shipped with two sets of 33 HTML files that you can use for online help files and source files for the `Help` database. The content of the HTML files addresses the default PLWeb Turbo interface, operators, and advisors. After you have configured PLWeb Turbo to perform to your specifications, you may want to edit the HTML files so they accurately represent to your users your customized PLWeb Turbo. The help files are located in `<drive>:\<install-Dir>\plweb\htdocs\help`.

If you edit one or more HTML files that are part of the `Help` database, you will need to recreate the database. For more information, refer to “Creating a Database” on page 91.

The following table alphabetically lists each HTML file.

Table 11-1: List of Online Help Files

File Name	Record/Screen Title
`about.html`	About this Online Help
`advexpl.html`	What is an Advisor
`advtool.html`	Advanced Searching Tools
`boolops.html`	Searching with Boolean Logic
`compqry.html`	Building a Complex Query
`conop.html`	Word Level Concept Searching
`consrch.html`	Concept Search: Exploring a General Idea
`dbdef.html`	Database Information
`dbfund.html`	Text Database Fundamentals

Table 11-1: List of Online Help Files

File Name	Record/Screen Title
defop.html	Understanding the Default Operator
dicadv.html	Viewing a Database's Dictionary
fld1op.html	Searching Specific Fields for a Query Word
fld2op.html	Restricting an Entire Query to Specific Fields
fldops.html	Combined Word-Level and Query-Level Field
fuzadv.html	Fuzzy Matching: Identifying Words with Similar Spelling
fuzop.html	Identifying Words with Similar Spelling
intsrch.html	Intelligent Searching
natlang.html	Natural Language Searching
nearops.html	Searching for Words that are Near Each Other
oltoc.html	Table of Contents
opexpl.html	What is a Query Operator?
precrul.html	Precedence Rules
ptfldnm.html	Using Partial Field Names
reladv.html	Finding Words Related to Your Query
relrank.html	Relevance Ranking
resscrn.html	Using the Search Results Screen
schscrn.html	Using the Search Screen
srchweb.html	Searching with PLWeb Turbo
valops.html	Searching for Fields with Values and Value Ranges
viewrec.html	Viewing Records
virtdb.html	Virtual Database: Searching Multiple Databases
wildops.html	Using Wildcards
xphrop.html	Searching for an Exact Phrase
xwdop.html	Searching for an Exact Word

11.6 More on Using HTML Frames

PLWeb Turbo supports HTML frames through its generic template model. In general, when using frames, one constructs a page containing the frameset and two or more pages containing the contents of the individual frames. In order to use HTML frames with PLWeb Turbo, you must create the frameset as a template, not as an HTML document. This frameset will become the target of an operation in PLWeb Turbo. One or more of the component frames in the frameset will be a URL interpreted by PLWeb Turbo.

Example: Creating a Combined Hitlist/Document Screen

As an example, assume that you wanted to create a document display screen consisting of two frames: one frame containing controls to move to the previous and next hit word in the document and one frame containing the document itself. First, replace the link in the hitlist template to point to the new frameset template; then, create a template containing the frameset.

Assume this example of a frameset template:

```
<FRAMESET COLS="20,80">
<FRAME name="control" SRC="/plweb/control.html">
<FRAME name="doc" SRC="{$DOCLINK: hrefonly=true, target=predoc.tmpl}">
</FRAMESET>
```

The first frame in this set is a regular HTML page. In this example, the page contains the controls for moving to the previous or next hit word in the document. Note that JavaScript support is required for this functionality, but not for the use of frames in general. The second frame contains a macro that PLWeb Turbo expands in order to retrieve the actual document. The `control.html` file would contain the following:

```
<BODY>
<FORM>
<INPUT TYPE=button VALUE="Next Hit" OnClick="parent.doc.goForward()"> <p>
<INPUT TYPE=button VALUE="Prev. Hit" OnClick="parent.doc.goBackward()"><p>
</FORM>
</BODY>
```

In addition to the actual document, the document template `predoc.tmpl` needs to contain the JavaScript for the `goForward()` and `goBackward()` functions. The following would be included in the `HEAD` section of the template:

```
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
<!--
var id = "-1"
function goForward() {
   id++
   location.hash = "hit" + id
}
function goBackward() {
   if (id > 0) {
      id--
   }
   location.hash = "hit" + id
}
// -->
</SCRIPT>
</HEAD>
```

Example: Creating a Go To Next Hit Word in Document Control Using HTML Frames

As a second example, assume that you wanted to create a screen that contains the hitlist in one frame and the selected document in the second frame. Again, you need to create a frameset as a template,

and point the target of the previous screen (the `prehit.tmpl` template or the `search.tmpl` template) to this template.

```
<FRAMESET COLS="30,70">
<FRAME name="hitlist" SRC="{$TOHITLIST: hrefonly=true, target=hitlist.tmpl}">
<FRAME name="doc" SRC="/plweb/empty.html">
</FRAMESET>
```

In this example, the first frame causes the hitlist to be displayed. The second screen initially points to an HTML page displaying an empty page; however, with a modified `hitlist.tmpl` file, the document links will cause the document to be displayed in the second frame:

```
{$HITLIST}
...
<A HREF={$DOCLINK: hrefonly=true, target=doc.tmpl} TARGET=doc>
...
{$HITLIST_END}
```


A. Configuring the Custom Library

The custom.dll Custom Library

PLWeb Turbo includes a shared library, `custom.dll`, that you can modify to add functionality to your application. Examples of changes you can make to the behavior of PLWeb Turbo by modifying the custom library include interface changes (addition of custom macros), adding logging, and adding automatic billing. The library contains a set of predefined functions that are called by PLWeb Turbo. Some of these functions, in turn, can call callback functions in PLWeb Turbo proper to access certain state information.

The interface to both the functions in the custom library and the callback functions is C. You can use C++ instead, but you must ensure that the functions expected by PLWeb Turbo are not mangled by the C++ compiler. If your C++ compiler predefines the `__cplusplus` macro, this will be handled automatically.

The custom library is located in the `<drive>:\<installDir>\plweb\custom` directory. To rebuild the custom library, refer to “Creating a Custom Macro” on page 162.Once the library has been rebuilt, you must copy the new shared library to the `<drive>:\<installDir>\plweb\lib` directory in order for PLWeb Turbo to find it. You must also restart the PLWeb Turbo daemon.

The following functions are part of the `custom.dll` library. If you want to customize PLWeb Turbo, you need to modify one or more of the following functions:

```
plwebInit()
plwebForkInit()
plwebShutdown()
plwebPreTemplate()
plwebPostTemplate()
plwebCustomMacro()
```

✉✉ **Note** ✉✉

The PLWeb Turbo daemon is linked against the custom library. As such, the custom library becomes part of the daemon process at runtime. If the custom library is defective or otherwise misbehaves, it is possible, even likely, that the operation of the PLWeb Turbo daemon will suffer. If you have customized the custom library and experience problems with PLWeb Turbo, especially severe problems like segmentation faults, or when the daemon does not appear to be servicing requests anymore, we recommend first replacing the custom version of the library with the shipping one to see if this solves the problem.

---

plwebInit()

```
int plwebInit(
    const char * lpszPlwebRootPath,         // Plweb root directory
    char * lpszBuf,                         // Error message to print
    size_t nBufLen );                       // Size of message buffer
```

Description The `plwebInit` function is called when the PLWeb Turbo daemon starts its initialization process. It can be used, for example, to allocate and initialize resources, read in data files or configuration files, initialize a link with a relational database, etc.

Arguments PLWeb Turbo will pass the full path to the PLWeb root directory to the function, so that it can dynamically determine its location (`lpszPlwebRootPath`). This is useful when accessing files that are installed relative to the PLWeb Turbo installation. It passes a buffer to the function which can be used to return an error message (`lpszBuf`). The size of the message should not exceed the size of the buffer
(`nBufLen`).

Return Values The function returns an integer to PLWeb Turbo. A return value of 0 indicates success. If a failure was encountered that should prevent PLWeb Turbo from completing its initialization, a non-zero value should be returned. The return value is assumed by PLWeb Turbo to be the error ID. If an error is returned, the function can return an error message in the buffer passed to it by PLWeb Turbo (`lpszBuf`).

plwebShutdown()

```
void plwebShutdown();
```

Description The `plwebShutdown` function is called during the PLWeb Turbo daemon shutdown process. The custom library should shutdown any resources that it needs to dispose of, and delete any memory it allocated in `plwebInit()`.

Arguments None.

Return Values None. The shutdown of PLWeb Turbo will proceed regardless of whether the custom library shutdown was successful.

plwebPreTemplate()

```
int plwebPreTemplate(
    void * lpState,       // State pointer
    char * lpszBuf,       // Error message to print
    size_t nBufLen );     // Size of message buffer
```

Description The `plwebPreTemplate` function is called once for every request, just before the PLWeb Turbo daemon starts processing the target template. Generally, you will use this function for activities that you want to occur before PLWeb Turbo processes and performs actions on the template.

The `plwebPreTemplate` function can access the callback functions (refer to “Callback Functions” on page 159) to access state information in PLWeb Turbo. PLWeb Turbo generates most of its state information while processing the template, which has not yet occurred when this function is called. Any state information that is accessed by this function, which is not yet present in the state, will be generated. This can cause additional processing time for each user request. If the same state information is also being used by the template, it will not need to be generated again. For example, if this function requests to know the number of hits generated for the search, a search may have to be executed in order to provide the answer. If the target template also contains a macro to display the number of hits, the search will not be executed again.

You can use this function, for example, to create a simple access log. Because most PLWeb Turbo activity takes place during template processing, this function will be called under all but a few error conditions. By logging the full request, the log can be used for diagnostic purposes and to resolve end-user problems.

Arguments The function receives a `void *` state pointer, which is needed when any of the PLWeb Turbo callback functions are accessed (`lpState`). It passes a buffer to the function that can be used to return an error message (`lpszBuf`). The size of the message should not exceed the size of the buffer (`nBufLen`).

Return Values The function returns an integer to PLWeb Turbo. A return value of 0 indicates success. If a failure was encountered that should prevent PLWeb Turbo from completing the request, a non-zero value should be returned. The return value is assumed by PLWeb Turbo to be the error ID. If an error is returned, the function can return an error message in the buffer passed to it by PLWeb Turbo (`lpszBuf`).

plwebPostTemplate()

```
void plwebPostTemplate(
    void * lpState );        // State pointer
```

Description The `plwebPostTemplate` function is called once for every request, just after the PLWeb Turbo daemon completes processing the target template. This function has the same access to state information as `plwebPreTemplate`. When the `plwebPostTemplate` function is called, however, all of the state information needed to process the template has already been generated and can be accessed from this function with very little expense. If you want to generate detailed access logs, you will typically use this function.

Arguments The function receives a `void *` state pointer that is needed when any of the PLWeb Turbo callback functions are accessed (`lpState`).

Return Values None. Processing of the request proceeds in PLWeb Turbo regardless of this function.

plwebCustomMacro()

```
int plwebCustomMacro(
    void * lpState,                 // State pointer
    const char * lpszMacroName,    // Macro name
    const char * lpszMacroArgs,    // Macro arguments
    char * lpszBuf,                // Error message / Result
    size_t nBufLen );              // Size of message buffer
```

Description Whenever PLWeb Turbo processes a template and finds a macro, it goes through a macro dispatcher that calls the correct handler for the macro encountered. If PLWeb Turbo fails to find a handler for the macro (i.e., if PLWeb Turbo does not recognize the macro), it will call the `plwebCustomMacro` function. This mechanism removes the need for registration of custom macros.

Since this function will be called for all custom macros, you will typically want to implement this function as a dispatcher, looking at the contents of the `lpszMacroName` buffer and calling the appropriate macro handler.

Arguments The function receives a `void *` state pointer that is needed when any of the PLWeb Turbo callback functions are accessed (`lpState`). It is also passed a buffer containing the name of the macro that was found (`lpszMacroName`). The macro is passed without the leading `{` and `$` characters and without the trailing `:` and `}` characters. If the macro contains arguments, the argument string is passed in the second buffer (`lpszMacroArgs`). It passes a buffer to the function that can be used to return the HTML generated by the custom macro, or an error message (`lpszBuf`). The size of the HTML or error message should not exceed the size of the buffer (`nBufLen`).

Return Values If the `plwebCustomMacro` function succeeds, it should return a 0 (zero) integer, and the result of the operation (the HTML code to be inserted into the output stream) should be stored in the `lpszBuf` buffer passed to it by PLWeb Turbo. If the function fails, a non-zero value should be returned, and the error message should be stored in the same buffer where the function would normally store its result.

Callback Functions

The `plwebPreTemplate()`, `plwebPostTemplate()`, and `plwebCustomMacro()` functions have access to data in PLWeb Turbo. There are two general callback functions available in PLWeb Turbo: one for getting information of type `char *` (string information), and one for getting information of type `long` (integer information).

getPlwebString()

```
int getPlwebString(
    void * lpState,                 // State pointer
    const char * lpszVarName,       // Variable to get
    const char * lpszVarArgs,       // Variable arguments
    char * lpszBuf,                 // Buffer to store var
    size_t nBufSize );              // Buffer size
```

Description This function retrieves a string from PLWeb Turbo by accessing its current state. The information may or may not be part of the state when it is requested. If the information is not currently present, PLWeb Turbo will attempt to generate the information.

Arguments You would pass this function the state pointer (`lpState`) that was passed to the calling function (`plwebPreTemplate()`, `plwebPostTemplate()`, or `plwebCustomMacro()`). All state information is accessed by asking for a specific variable (`lpszVarName`). Table 11-2, “Callback Function Variables,” on page 160 shows a list of valid variable names. If additional arguments are required for a variable, they are passed as a separate argument (`lpszVarArgs`). Most variables do not currently take arguments, in which case you simply pass `NULL`. You also provide a `char *` buffer in which you want the requested string to be stored (`lpszBuf`). The `getPlwebString()` function guarantees that it will not exceed the length of the passed buffer (`nBufSize`).

Return Values The function returns either 0 on success or a non-zero error code otherwise.

getPlwebLong()

```
int getPlwebLong(
    void * lpState,                 // State pointer
    const char * lpszVarName,      // Variable to get
    const char * lpszVarArgs,      // Variable arguments
    long * lpLongVal );            // Value to get
```

Description This function retrieves an integer (long) value from PLWeb Turbo by accessing its current state. The information may or may not be part of the state when it is requested. If the information is not currently present, PLWeb Turbo will attempt to generate the information.

Arguments You would pass this function the state pointer (`lpState`) that was passed to the calling function (`plwebPreTemplate()`, `plwebPostTemplate()`, or `plwebCustomMacro()`). All state information is accessed by asking for a specific variable (`lpszVarName`). Table 11-2, "Callback Function Variables," on page 160 shows a list of valid variable names. If additional arguments are required for a variable, they are passed as a separate argument (`lpszVarArgs`). Most variables do not currently take arguments, in which case you simply pass `NULL`. You also provide a pointer to a long (`lpLongVal`). The function will store the result in the address provided.

Return Values The function returns either 0 on success or a non-zero error code otherwise.

Valid Variable Names

The state information is accessed by specifying the name of the variable in the `lpszVarName` buffer. The following table lists the variables that can be accessed.

> ✉✉ **Note** ✉✉
>
> The names that are passed to the callback functions are case sensitive.

Table 11-2: Callback Function Variables

Name	Retrieves	Type
`Request.User`	Returns the user name. This value is available only if HTTP daemon authentication is enabled.	String
`Request.UserAgent`	Returns a string identifying the user's browser. Usually, one can infer browser type (Netscape, Internet Explorer, etc.), browser version, and OS information from this string.	String
`Request.SearchOp`	Returns the default search operator (`AND`,`OR`, `ADJ`) that is defined in the search screen template.	String
`Request.CustomLogString`	Returns the custom log string that either you hardcoded or the end user chose. For further information, refer to the `CUSTOMLOGSTRING` macro on page 194.	String
`Request.Sorting`	Returns the sorting method that either you hardcoded or the end user chose. Values that can be returned are `none`, `byrelevance`, and `byfield:[+|-]fields`. For further information, refer to "search_impl.tmpl" on page 172.	String

Table 11-2: Callback Function Variables

Name	Retrieves	Type
`Request.TemplateName`	Returns the name of the template the user is accessing to service the request.	String
`Request.SelectedDatabases`	Returns the list of databases the user has selected. The databases are separated by commas.	String
`Request.Query`	Returns the non-fielded query the user entered.	String
`Request.Query.<N>`	Returns the fielded query component with ID `<N>`.	String
`Request.MaxItems`	Returns the maximum number of items the user requested to have returned.	Long
`Request.StartHit`	Returns the rank of the first hit the user requested to have returned.	Long

`Request.QueryType`	Returns the type of query the user is executing. The return value will be one of the following:
	
	`0 simple search`
	`1 concept search`
	`2 relate advisor`
	`3 fuzzy (spelling) advisor`
	`4 dictionary advisor`
		Long

`Document.DBName`	Returns the name of the database to which the current document belongs.	String
`Document.Class`	The record class of the current document. For a list of record classes, refer to the `IF` macro's `DocType` variable on page 205.	String
`Document.RemoteURL`	Returns the URL of the current document, if the document is a remote document.	String
`Document.ID`	Returns the document ID of the current document.	Long
`Document.Rank`	Returns the rank of the current document in the current hitlist.	Long
`Document.Price`	Returns the contents of the `PRICE` field, if it is defined for the current document.	Long
`Document.Score`	Returns the raw score of the current document in the current hitlist.	Long
`Document.Size`	Returns the size of the current document in characters. This is a character count, not a byte count.	Long
`Hitlist.ItemsFound`	Returns the number of items found for the current search.	Long
`Hitlist.TopScore`	Returns the raw score of the top ranked document in the current hitlist. This is only valid when sorting by relevance.	Long
`View.Name`	Returns the name of the current view	String
`View.Location`	Returns the full path to the current view.	String
`CGI.<VAR>`	Returns the value of the CGI variable as defined in the HTTP environment.	String

Creating a Custom Macro

To add a custom macro MYMACRO to the library, perform the following steps (assuming a C implementation).

1. Open Microsoft Visual C++ 5.0. (This is the compiler we used to build PLWeb Turbo. We do not guarantee that PLWeb Turbo will work with another compiler.)

2. In Microsoft Visual C++, select OPEN WORKSPACE from the File menu.

3. Open <drive>:\<installDir>plweb\custom*.dsp.

4. From the Tools menu, select OPTIONS.

5. In the Options window, click the DIRECTORIES tab.

6. In the Show Directories scroll box, select LIBRARY FILES.

7. Add <drive>:\<installDir>\lib.

8. Add a new test in the plwebCustomMacro() function to test for the new MYMACRO macro.

   ```
   else if (strcmp(lpszMacroName, "MYMACRO") == 0)
   ```

   If the implementation of the macro is simple, you can add it in place. If not, create a new function macroMYMACRO(), and call it from this dispatch function.

9. From the Build menu, select REBUILD ALL.

10. Add the new MYMACRO macro to one of the templates.

11. Backup the original custom library located in <drive>:\<installDir>\plweb\lib\custom.dll.

12. Copy the new library from the <drive>:\<installDir>\plweb\custom directory to the <drive>:\<installDir>\plweb\lib\Release directory.

13. Restart the PLWeb Turbo daemon.

A Custom Billing Application

As an example of what can be implemented using the custom library, PLWeb Turbo provides a sample billing library. You can enhance the billing functionality to satisfy most billing needs. The shipping version will allow you to create a global billing table that specifies the document price for documents from a database. Different document prices can be specified for different databases, but all documents within a database will have the same specified price in the table. This does not mean, however, that individual documents cannot have prices different from the price in the table. The PRICE field (determined by the PRICE macro, which is discussed on page 214) in documents is still supported and will override any setting in the table.

Price information is stored in <drive>:\<installDir>\plweb\etc\price.conf.

Format

The format of the `price.conf` file is as follows:

```
<dbname>   <price>
```

The following table explains the variables you must use when defining the `price.conf` file:

Table 11-3: price.conf file

Argument	Description	Explanation
<dbname>	Database Name	Specifies the name of the database for which you are defining pricing. You may only have one database per line, and <dbname> must be separated from <price> by a tab (not spaces).
<price>	Price in Units	Specifies the price, in units (cents), for each document in the database.

Example

```
Travel     125
News       75
Help       25
```

In the U. S., this would correspond to $1.25 for each document from the `Travel` database, $0.75 for each document from the `News` database, and $0.25 for each document from the `Help` database.

The billing functionality can be changed to accommodate different scenarios. For example, you can enhance the billing table as follows:

- To allow prices based on the length of the document, use the document size information by asking for the callback function `Document.Size` (refer to Table 11-2 on page 160).
- To allow different price tables for each view, use the view location information by asking for the callback function `View.Location` (refer to Table 11-2 on page 160).

Overriding the Database-wide Pricing

You can override a database's pricing on a document-by-document basis by including the `PRICE` field in your field markup (refer to page 55). If a particular document includes a `PRICE` field, PLWeb Turbo will read the contents of that field, and the value read will override the settings in the `price.conf` file. In other words, the following document in the `News` database would cost only $0.50, as opposed to $0.75:

```
-PRICE-
50
-TEXT-
... document content ...
-END-
```


Displaying Document Price to the End User

To display the price for a document to the user, you must use the CPRICE (custom price) macro (discussed on page 193). This macro will calculate and either display or log the price of the current document based on the contents of the table and the contents of the PRICE field as discussed above.

Further, this macro can also be used to log the price entry to a billing file. For example

```
...
{$OUTPUT: target=billing_log}
Host: {$CGIVAR: name=REMOTE_HOST}, User: {$CGIVAR: name=REMOTE_USER}
Database: {$DBNAME}, Document ID: {$DOCID}, Price: {$CPRICE}
{$OUTPUT: target=DISPLAY}
...
```


B. Understanding the Default Templates

The following table summarizes the function of all of the default templates.

Table 11-4: Default Templates

Template Name	Description
dict.tmpl fuzzy.tmpl relate.tmpl	Advisor framework templates. Comprised of the HTML code that creates the basic structure of the advisor screens. Included by prehit.tmpl when the user elects to do an advisor search.
dict_impl.tmpl fuzzy_impl.tmpl relate_impl.tmpl	Advisor implementation templates. Contain the PLWeb Turbo macros associated with each advisor search. Included in the respective *.tmpl template via the INCLUDE macro.
doc.tmpl doc_html.tmpl	ASCII and HTML document-display framework templates. Comprised of the HTML code that creates the basic structure of the ASCII and HTML document display screens, respectively. Included by predoc.html when the user clicks on a hit in the hitlist in order to display a document.
doc_impl.tmpl doc_html_impl.tmpl	ASCII and HTML document-display implementation templates. Contain the PLWeb Turbo macros associated with displaying ASCII and HTML documents. Included in the doc.tmpl or doc_html.tmpl templates via the INCLUDE macro.
error.tmpl	Error display framework template. Comprised of the HTML code that creates the basic structure of the error screen. Included by PLWeb Turbo when an error occurs while PLWeb Turbo is not actively processing a template.
error_impl.tmpl	Error display implementation template. Contains the PLWeb Turbo macros associated with displaying error messages. Included in the error.tmpl template via the INCLUDE macro. Also included in almost all templates to display error messages when an error occurs while a template is being processed.
hitlist.tmpl	Hitlist display framework template. Comprised of the HTML code that creates the basic structure of the hitlist screen. Included by prehit.tmpl when the user elects to do a simple search (rather than an advisor search).
hitlist_impl.tmpl	Hitlist display implementation template. Contains the PLWeb Turbo macros associated with generating the hitlist. Included in the hitlist.tmpl template via the INCLUDE macro.
logo.tmpl	Logo display template. Contains the HTML code that creates the logo that is common to all of the templates. Included in all templates via the INCLUDE macro.
predoc.tmpl	Template selector for document display. Relay template included by hitlist_impl.tmpl to determine which document display template should be included based on the type of document selected (doc.tmpl for ASCII or doc_html.tmpl for HTML).
prehit.tmpl	Template selector for hitlist or advisor searches. Relay template included by search_impl.tmpl to determine which template should be displayed based on the type of action the user performed (hitlist.tmpl for simple searches or relate.tmpl, fuzzy.tmpl or dict.tmpl for advisor searches).
search.tmpl	Search screen framework template. Comprised of the HTML code that creates the basic structure of the search screen. Included by views_impl.tmpl after the user has selected a view.
search_impl.tmpl	Search screen implementation template. Contains the PLWeb Turbo macros associated with generating the search screen. Included in the search.tmpl template via the INCLUDE macro.
views.tmpl	View selection screen framework template. Comprised of the HTML code that creates the basic structure of the view selection screen. Initial screen displayed by PLWeb Turbo.

Table 11-4: Default Templates

Template Name	Description
views_impl.tmpl	View selection screen implementation template. Contains the PLWeb Turbo macros associated with generating the view selection screen. Included in the views.tmpl template via the INCLUDE macro.

The following code and explanatory text is found in all default templates for the PLWeb Turbo sample application. The templates are presented in roughly the same order in which the user would traverse them in a typical user session. This section will be a useful reference when implementing similar functionality in your PLWeb Turbo application.

views.tmpl

✉✉ Note ✉✉

The default view selection screen template is implemented such that, when users have access to only one view, they will not see the view selection screen; they will go directly to the search screen. You may change this at any time by removing the first three lines of the view.tmpl template.

```
<!--The first two lines will cause PLWeb Turbo to go directly to the search
screen if only one view exists. This line tells PLWeb Turbo that, if the NUM-
VIEWS variable is equal to one, go to the search.tmpl template. This is the
first example of using the INCLUDE macro to separate complex functionality from
framework. Rather than embed the code for the search screen template, this
macro allows us to separate the functionality into another file.-->

{$IF: variable=NUMVIEWS, operator=EQ, value=1}
{$INCLUDE: file=search.tmpl}

<!--In all other cases, if no views or more than one view exists, proceed with
displaying the view selection screen template. The next section is standard
HTML that defines the look and feel of the view selection screen.-->

{$ELSE}

<HTML>

<HEAD>
<TITLE>PLWeb View Selection</TITLE>
</HEAD>

<BODY BGCOLOR="#ffffff" BACKGROUND="/plweb/icons/subback.jpg" TEXT="#001572"
LINK="#008800" ALINK="#000000" VLINK="#bb4400">

<A NAME="top"></A>
```

```
<TABLE BORDER="0" CELLSPACING="0" CELLPADDING="0" WIDTH="600">

<TR>

<!--The following line causes the contents of logo.tmpl to be inserted into the
template. The logo template was centralized into a separate template so that
the HTML that creates the look and feel was not repeated in every template.-->

{$INCLUDE: file=logo.tmpl}

<TD VALIGN="TOP">
    <IMG SRC="/plweb/icons/view_word.gif" WIDTH="267" HEIGHT="48" BORDER="0"
    ALT="View Selection" NATURALSIZEFLAG="0" ALIGN="BOTTOM"></TD>
<TD VALIGN="TOP">
    <IMG SRC="/plweb/icons/fillline.gif" WIDTH="173" HEIGHT="48" BORDER="0"
    ALT="-----" NATURALSIZEFLAG="0" ALIGN="BOTTOM"></TD>
<TD VALIGN=TOP ROWSPAN="2"><IMG SRC="/plweb/icons/viewselect_icon.gif"
HEIGHT=104 WIDTH=102></TD>
</TR>

<!--The next line causes PLWeb Turbo to read the view selection screen imple-
mentation template (views_impl.tmpl). The view selection screen template is
divided into a separate template to distinguish between the look-and-feel
aspects of the view selection screen and the actual PLWeb Turbo functionality
of the screen.-->

{$INCLUDE: file=views_impl.tmpl}

<TR>
<TD WIDTH="17%"> </TD>
<TD WIDTH="17%"> </TD>
<TD WIDTH="17%"> </TD>
<TD WIDTH="17%"> </TD>
</TR>

</TABLE>

</BODY>
</HTML>
{$ENDIF}
```

views_impl.tmpl

```
<TR>
<TD COLSPAN="2"><P>
<IMG SRC="/plweb/icons/sp_1_56.gif" WIDTH="400" HEIGHT="20" NATURALSIZE-
FLAG="0" ALIGN="BOTTOM"><br>

<!--In the event that no views are available—a condition that only an adminis-
trator should encounter during preliminary implementation and testing—the next
```

```
two lines will cause the message “No views are currently available” to be sent
to the browser for display.-->

{$IF: variable=NUMVIEWS, operator=EQ, value=0}
    <H3><FONT COLOR="0bb0b0">No views are currently available</FONT></H3>

<!--Otherwise, when multiple views are available, the remainder of this screen
will create a radio-button list of the available views for the user to select.-
->

{$ELSE}

<!--The following line marks the beginning of an HTML form.-->

<FORM METHOD=POST ACTION="http:fastweb">

<!--The following line is the first example of defining the target of this tem-
plate. When the form is submitted, PLWeb Turbo will proceed to the search.tmpl
template. When a template contains a form, the reference to the target template
should be included within the form. For purposes of continuity, we place the
target immediately following the beginning of the form.-->

<INPUT TYPE=HIDDEN NAME=TemplateName VALUE="search.tmpl">

<B>Select the view you want to use: </B>

<P>
{$VIEWLIST}
    <input type=radio name=view value={$VIEWNAME}>{$VIEWDESC}<br>
{$VIEWLIST_END} <BR>

<INPUTTYPE=IMAGENAME=submitSRC="/plweb/icons/continue-.gif"WIDTH="127"HEIGHT="28"
BORDER="0" ALT="Continue" NATURALSIZEFLAG="0" ALIGN="BOTTOM">

</FORM>

<HR ALIGN=LEFT>

{$ENDIF}

<!--The next three lines will cause the error template to be included if an
error is encountered while processing this template.-->

{$IF: variable=error, operator=EQ, value=TRUE}
    {$INCLUDE: file=error_impl.tmpl}
{$ENDIF}

</TD>
</TR>
```


search.tmpl

```
<!--This template primarily establishes look and feel.-->

<HTML>

<HEAD>
    <TITLE>PLWeb Turbo Search</TITLE>
</HEAD>

<BODY BGCOLOR="#ffffff" BACKGROUND="/plweb/icons/subback.jpg" TEXT="#001572"
    LINK="#008800" ALINK="#000000" VLINK="#bb4400">

<TABLE BORDER="0" CELLSPACING="0" CELLPADDING="0" WIDTH="600">

<TR>
{$INCLUDE: file=logo.tmpl}

<TD VALIGN="TOP">
    <IMG SRC="/plweb/icons/search_word.gif" WIDTH="267" HEIGHT="48" BORDER="0"
        ALT="Search" NATURALSIZEFLAG="0" ALIGN="BOTTOM"></TD>
<TD VALIGN="TOP">
    <IMG SRC="/plweb/icons/fillline.gif" WIDTH="173" HEIGHT="48" BORDER="0"
        ALT="-----" NATURALSIZEFLAG="0" ALIGN="BOTTOM"></TD>

<TD VALIGN=TOP ROWSPAN="2"><IMG SRC="/plweb/icons/search_icon.gif" HEIGHT=72
WIDTH=102></TD>
</TR>

<!--The following line causes the search screen implementation template to be
included.-->

{$INCLUDE: file=search_impl.tmpl}

<TR>
<TD WIDTH="17%"> </TD>
<TD WIDTH="17%"> </TD>
<TD WIDTH="17%"> </TD>
<TD WIDTH="17%"> </TD>
</TR>

</TABLE>

</BODY>
</HTML>
```

search_impl.tmpl

```
<TR>
<TD COLSPAN="2">
```

```
<P> <IMG SRC="/plweb/icons/sp_1_56.gif" WIDTH="400" HEIGHT="20" NATURALSIZE-
FLAG="0" ALIGN="BOTTOM"><br>

<B>Select the databases you want to search.</B>
<A HREF="/plweb/dbdef.html">[ Database descriptions ]</A><BR>

</TD>
</TR>

<TR>
<TD COLSPAN=3>

<FORM METHOD=POST ACTION="http:fastweb" ENCTYPE="x-www-form-encoded">

<!--The following line defines the target of this template.-->

<input type=hidden name=TemplateName value="prehit.tmpl">

<input type=hidden name=view value="{$VIEW}">

<!--The next five lines create the list of databases available in the current
view. The HTML creates a list of check boxes and pre-checks any databases that
the user has selected throughout the session.-->

{$DBLIST}
<input type=checkbox {$IF: variable=SELECTEDDBS, operator=CONTAINS,
value={$DBNAME}checked{$ENDIF}
name=dbname value="{$DBNAME}">{$DBNAME}
{$DBLIST_END}

<P>

<B>Describe what you are looking for. </B>

<!--These three lines create the query box and pass the query via the QUERY
macro to the query rule for processing.-->

<A HREF="/plweb/operat.html">[ Searching tips ]</A><BR>
<INPUT TYPE="text" NAME="query" SIZE="45" value="{$QUERY}">

<!--The next line establishes the query rule.-->

<input type = hidden name=query_rule value="($QUERY)">

<P>

<!--The next several lines—between <SELECT NAME="numresults"> and </SELECT>—
create a drop-down menu of choices with 25 pre-selected. By enclosing the menu
in nested IF statements, the values selected by the user will be preserved when
he/she returns to this page during a user session.-->
```



```
<B>Number of results </B>
<SELECT NAME="numresults">
    {$IF: variable=MAXITEMS, operator=EQ, value=10}<option selected> 10
{$ELSE}<option> 10 {$ENDIF}
    {$IF: variable=MAXITEMS, operator=EQ, value=25}<option selected> 25 {$ELSE}
        {$IF: variable=MAXITEMS, operator=EQ, value=0}<option selected> 25
{$ELSE}<option> 25 {$ENDIF}
    {$ENDIF}
    {$IF: variable=MAXITEMS, operator=EQ, value=50}<option selected> 50
{$ELSE}<option> 50 {$ENDIF}
    {$IF: variable=MAXITEMS, operator=EQ, value=100}<option selected> 100
{$ELSE}<option> 100{$ENDIF}
</SELECT>.

<!--The following lines—between <SELECT NAME="sorting"> and </SELECT>—create a
drop-down menu of sorting choices with Relevance pre-selected. As with the num-
ber of results menu above, these options will be preserved in the session.-->

<B>Sorting </B>
<SELECT NAME="sorting">
    {$IF: variable=SORTING, operator=EQ, value=byrelevance}<option selected
value="byrelevance"> Relevance {$ELSE}<option value="byrelevance"> Relevance
{$ENDIF}
    {$IF: variable=SORTING, operator=EQ, value=none}<option selected
value="none"> No Sorting {$ELSE}<option value="none"> No Sorting {$ENDIF}
</SELECT>.

<P>
<TABLE WIDTH="65%" BORDER="0" CELLSPACING="2" CELLPADDING="0">

<!--The following section—from the next line until the end of the table—inserts
image links to select between several search options.-->

<TR>
<TD><INPUT TYPE=IMAGE NAME=simplesearch SRC="/plweb/icons/search-.gif"
WIDTH="96" HEIGHT="28" BORDER="0" ALT="Search" NATURALSIZEFLAG="0" ALIGN="BOT-
TOM"></TD>
<TD><INPUT TYPE=IMAGE NAME=concept SRC="/plweb/icons/conceptsc-.gif"
WIDTH="96" HEIGHT="28" BORDER="0" ALT="Concept Search" NATURALSIZEFLAG="0"
ALIGN="BOTTOM"></TD>
<TD><INPUT TYPE=IMAGE NAME=relate SRC="/plweb/icons/relate-.gif" WIDTH="96"
HEIGHT="28" BORDER="0" ALT="Relate" NATURALSIZEFLAG="0" ALIGN="BOTTOM"></TD>
<TD><INPUT TYPE=IMAGE NAME=fuzzy SRC="/plweb/icons/spelling-.gif" WIDTH="96"
HEIGHT="28" BORDER="0" ALT="Spelling" NATURALSIZEFLAG="0" ALIGN="BOTTOM"></TD>
<TD><INPUT TYPE=IMAGE NAME=dictionary SRC="/plweb/icons/dictionary-.gif"
WIDTH="96" HEIGHT="28" BORDER="0" ALT="Dictionary" NATURALSIZEFLAG="0"
ALIGN="BOTTOM"></TD>
</TR>

</TABLE>
```

```
<HR ALIGN=LEFT>
| <A HREF="/plweb/help/oltoc.html">Online help</A>
| <A HREF="/plweb/index.html">Return to home page</A>

{$IF: variable=error, operator=EQ, value=TRUE}
    {$INCLUDE: file=error_impl.tmpl}
{$ENDIF}

</FORM>
</TD>
</TR>
```

prehit.tmpl

```
<!--This template establishes which template will be included next, based on
the user's selection.-->

<!--If the user selected DICTIONARY from the list of search options on
search.tmpl, the dict.tmpl template would be included.-->

{$IF: variable=QUERYTYPE, operator=EQ, value=DICTIONARY}
    {$INCLUDE: file=dict.tmpl}

<!--If the user selected RELATE, relate.tmpl would be included.

{$ELSE}{$IF: variable=QUERYTYPE, operator=EQ, value=RELATE}
    {$INCLUDE: file=relate.tmpl}

<!--If the user selected FUZZY, fuzzy.tmpl would be included.-->

{$ELSE}{$IF: variable=QUERYTYPE, operator=EQ, value=FUZZY}
    {$INCLUDE: file=fuzzy.tmpl}

<!--If the user does either a search or a concept search—because both proceed
to the same template—the hitlist.tmpl template would follow.-->

{$ELSE}
    {$INCLUDE: file=hitlist.tmpl}
{$ENDIF}
{$ENDIF}
{$ENDIF}
```

relate.tmpl

```
<!--This template is primarily standard HTML. It includes logo.tmpl to insert
common look and feel, and it includes relate_impl.tmpl to insert the functional
aspects of doing a related word search.-->
```


✉✉ Note ✉✉

The advisor templates—dictionary, relate, and fuzzy—are all functionally identical. The only differences are the wording and the graphics used. Therefore, we have only included a single advisor as an example in this section.

```
<HTML>

<HEAD>
  <TITLE>PLWeb Document</TITLE>
</HEAD>

<BODY BGCOLOR="#ffffff" BACKGROUND="/plweb/icons/subback.jpg" TEXT="#001572"
LINK="#008800" ALINK="#000000" VLINK="#bb4400">

<A NAME="top"></A>

<TABLE BORDER="0" CELLSPACING="0" CELLPADDING="0" WIDTH="600">

<TR>
{$INCLUDE: file=logo.tmpl}

<TD VALIGN="TOP">
  <IMG SRC="/plweb/icons/relate_word.gif" WIDTH="267" HEIGHT="48" BORDER="0"
    ALT="Relate" NATURALSIZEFLAG="0" ALIGN="BOTTOM"></TD>
<TD VALIGN="TOP">
  <IMG SRC="/plweb/icons/fillline.gif" WIDTH="173" HEIGHT="48" BORDER="0"
    ALT="-----" NATURALSIZEFLAG="0" ALIGN="BOTTOM"></TD>
<TD VALIGN=TOP ROWSPAN="2"><IMG SRC="/plweb/icons/relate_icon.gif" HEIGHT=104
WIDTH=102></TD>
</TR>

{$INCLUDE: file=relate_impl.tmpl}

<TR>
<TD WIDTH="17%"> </TD>
<TD WIDTH="17%"> </TD>
<TD WIDTH="17%"> </TD>
<TD WIDTH="17%"> </TD>
</TR>

</TABLE>

</BODY>
</HTML>
```

relate_impl.tmpl

```
<TR>
<TD COLSPAN="2"><P>
<IMG SRC="/plweb/icons/sp_1_56.gif" WIDTH="400" HEIGHT="12" NATURALSIZE-
FLAG="0" ALIGN="BOTTOM"><br>

<!--The following line creates a menu item at the top of the screen that links
back to the search page. The target of this link is search.tmpl.-->

| {$TOSEARCHFORM: target=search.tmpl}Search Screen{$EOL}
| <br>

<IMG SRC="/plweb/icons/fillline.gif" WIDTH="440" HEIGHT="1" BORDER="0"

    ALT="-----"NATURALSIZEFLAG="0" ALIGN="BOTTOM">

<IMG SRC="/plweb/icons/sp_1_56.gif" WIDTH="400" HEIGHT="16" NATURALSIZE-
FLAG="0" ALIGN="BOTTOM"><br>

</TD>
</TR>

<TR>
<TD COLSPAN=3>

<!--Given that at least one term is returned for the related word search, the
following section of the template will display a scrolling list from which the
user can select additional terms to be included in the query.-->

{$IF: variable=termsfound, operator=GT, value=0}

<FORM METHOD=POST ACTION="http:fastweb">
<INPUT TYPE=HIDDEN NAME=TemplateName value=search.tmpl>

{$HIDDENVALS}

<TABLE BORDER="0" CELLSPACING="0" CELLPADDING="0" WIDTH="400">
<TR>

<TD ROWSPAN="1" VALIGN="TOP" WIDTH="200">
<B>Related words for query: <BR>
<FONT COLOR=ff0000> {$QUERY} </FONT></B>

<IMG SRC="/plweb/icons/sp_1_56.gif" WIDTH="200" HEIGHT="52" NATURALSIZE-
FLAG="0" ALIGN="BOTTOM"><br>

Select words in the table, <BR>
then click here to <BR>
<IMG SRC="/plweb/icons/sp_1_56.gif" WIDTH="200" HEIGHT="10" NATURALSIZE-
FLAG="0" ALIGN="BOTTOM"><br>
<INPUT TYPE=IMAGE NAME=addwords SRC="/plweb/icons/addwords-.gif" WIDTH="125"
```



```
HEIGHT="27"
    BORDER=0 ALT="Add Words to Query" NATURALSIZEFLAG=0 ALIGN=BOTTOM>
</TD>

<TD ROWSPAN="1" VALIGN="TOP" WIDTH="200">
<SELECT NAME=list_entry MULTIPLE SIZE=10>

<!--The following section of code will cause the original query terms to be
highlighted.-->

{$RELATELIST}
{$IF: variable=QUERY, operator=CONTAINS, value={$RELATETERM}
    <option selected value={$RELATETERM}> {$RELATETERM}
{$ELSE}
    <option value={$RELATETERM}> {$RELATETERM}
{$ENDIF}
{$RELATELIST_END}

</SELECT>
</TD>

</TR>
</TABLE>

</FORM>
<HR ALIGN=LEFT>

<!--If no related terms are returned (but not as a result of an error), the
following two lines will send the message “No terms found for query” to the
browser to be displayed on the screen.-->

{$ELSE}
    {$IF: variable=error, operator=EQ, value=FALSE}
        <B>No terms found for query: <FONT COLOR=ff0000>{$QUERY}</FONT></B>

<!--However, if no terms were returned as the result of an error, the error
template will be included.-->

    {$ELSE}
        {$INCLUDE: file=error_impl.tmpl}
    {$ENDIF}
{$ENDIF}

</TD>
</TR>
```

hitlist.tmpl

```
<!--This template is predominantly standard HTML.-->
```

```
<HTML>

<HEAD>
    <TITLE>PLWeb Query Results</TITLE>
</HEAD>

<BODY BGCOLOR="#ffffff" BACKGROUND="/plweb/icons/subback.jpg" TEXT="#001572"
LINK="#008800" ALINK="#000000" VLINK="#bb4400">

<A NAME="top"></A>

<TABLE BORDER="0" CELLSPACING="0" CELLPADDING="0" WIDTH="600">

<TR>
{$INCLUDE: file=logo.tmpl}

<TD VALIGN="TOP">
    <IMG SRC="/plweb/icons/searchresult_word.gif" WIDTH="267" HEIGHT="48" BOR-
DER="0"
        ALT="Search Results" NATURALSIZEFLAG="0" ALIGN="BOTTOM"></TD>
<TD VALIGN="TOP">
    <IMG SRC="/plweb/icons/fillline.gif" WIDTH="173" HEIGHT="48" BORDER="0"
        ALT="------" NATURALSIZEFLAG="0" ALIGN="BOTTOM"></TD>
<TD VALIGN=TOP ROWSPAN="2"><IMG SRC="/plweb/icons/searchresult_icon.gif"
HEIGHT=104 WIDTH=102></TD>
</TR>

{$INCLUDE: file=hitlist_impl.tmpl}

<TR>
<TD WIDTH="17%"> </TD>
<TD WIDTH="17%"> </TD>
<TD WIDTH="17%"> </TD>
<TD WIDTH="17%"> </TD>
</TR>

</TABLE>

</BODY>
</HTML>
```

hitlist_impl.tmpl

```
<TR>
<TD COLSPAN="2"><P>
<IMG SRC="/plweb/icons/sp_1_56.gif" WIDTH="400" HEIGHT="12" NATURALSIZE-
FLAG="0" ALIGN="BOTTOM"><br>

<!--The next four lines create menu items that allow the user to either return
to the search screen or traverse the hitlist.-->
```



```
| {$TOSEARCHFORM: target=search.tmpl}Search Screen{$EOL}
| {$PREVLIST: target=hitlist.tmpl}Previous{$EOL}
| {$NEXTLIST: target=hitlist.tmpl}Next{$EOL}
| <br>

<IMG SRC="/plweb/icons/fillline.gif" WIDTH="440" HEIGHT="1" BORDER="0"
    ALT="-----"NATURALSIZEFLAG="0" ALIGN="BOTTOM">

<IMG SRC="/plweb/icons/sp_1_56.gif" WIDTH="400" HEIGHT="16" NATURALSIZE-
FLAG="0"ALIGN="BOTTOM"><br>
</TD>
</TR>

<TR>
<TD COLSPAN=3>

<!--Given the search results in at least one document, a statement of the num-
ber of results and the rank of those results will be sent to the browser for
display. Refer to the section on macros for more information on the {$HITS-
FOUND}, {$FIRSTHITRANK}, and {$LASTHITRANK} macros.-->

{$IF: variable=itemsfound, operator=GT, value=0}

<B> {$HITSFOUND} documents found for query:
<FONT COLOR=ff0000>{$QUERY}</FONT></B> <br>

Displaying documents {$FIRSTHITRANK} - {$LASTHITRANK} <br>

<TABLE BORDER=0 CELLSPACING=8>

<TR ALIGN=LEFT>
<TD><B>Score</B></TD>
<TD><B>Document Title</B></TD>
<TD><B>Database</B></TD>
<TD><B>Size</B></TD>
</TR>

<!--The next section—between {$HITLIST} and {$HITLIST_END}—creates the loop
that displays every hit in the hitlist. This code will be traversed for every
hit until the list of hits is complete. It will display the score, title, data-
base name, and document size in a table.-->

{$HITLIST}
<TR ALIGN=LEFT VALIGN=TOP>
<TD>{$SCORE: size=5}</TD>

<!--The target of the link to the document is predoc.tmpl, because a relay tem-
plate is required here. predoc.tmpl is the template that determines which doc-
ument template should be displayed—ASCII or HTML.-->

<TD>{$DOCLINK: target=predoc.tmpl}{$TITLE: lines=1, default=*** UNTI-
TLED***}{$EOL}<br>
```

```
    {$TITLE: lines=2, firstline=2}</TD>
<TD>{$DBNAME: size=12}</TD>
<TD>{$DOCSIZE: size=5}</TD>
</TR>
{$HITLIST_END}

</TABLE>

{$ELSE}

    {$IF: variable=error, operator=EQ, value=FALSE}
        <B>No hits found for query: <FONT COLOR=ff0000>{$QUERY}</FONT></B>
    {$ELSE}
        {$INCLUDE: file=error_impl.tmpl}
    {$ENDIF}

{$ENDIF}

</TD>
</TR>
```

predoc.tmpl

```
<!--This template determines which document template to include.-->

<!--If the document being retrieved requires the HTML reader, the HTML document
display template, doc_html.tmpl will be included.-->

{$IF: variable=DOCTYPE, operator=EQ, value=CplPlsHTMLReader}
    {$INCLUDE: file=doc_html.tmpl}

<!--Otherwise, for any other document type, the standard document display tem-
plate will be included.-->

{$ELSE}
    {$INCLUDE: file=doc.tmpl}
{$ENDIF}
```

doc.tmpl

✉✉ **Note** ✉✉

`doc.tmpl` and `doc_html.tmpl` are functionally identical but simply include different document display templates. Therefore, we have only included `doc.tmpl` as a reference.

```
<HTML>

<HEAD>
    <TITLE>PLWeb Document</TITLE>
</HEAD>

<BODY BGCOLOR="#ffffff" BACKGROUND="/plweb/icons/subback.jpg" TEXT="#001572"
LINK="#008800" ALINK="#000000" VLINK="#bb4400">

<A NAME="top"></A>

<TABLE BORDER="0" CELLSPACING="0" CELLPADDING="0" WIDTH="600">

<TR>
{$INCLUDE: file=logo.tmpl}

<TD VALIGN="TOP">
    <IMG SRC="/plweb/icons/document_word.gif" WIDTH="267" HEIGHT="48" BOR-
DER="0"
        ALT="Document" NATURALSIZEFLAG="0" ALIGN="BOTTOM"></TD>
<TD VALIGN="TOP">
    <IMG SRC="/plweb/icons/fillline.gif" WIDTH="173" HEIGHT="48" BORDER="0"
        ALT="-----" NATURALSIZEFLAG="0" ALIGN="BOTTOM"></TD>
<TD VALIGN=TOP ROWSPAN="2"><IMG SRC="/plweb/icons/document_icon.gif"
HEIGHT=104WIDTH=102></TD>
</TR>

<!--The next line includes the standard document display template.-->

{$INCLUDE: file=doc_impl.tmpl}

<TR>
<TD WIDTH="17%"> </TD>
<TD WIDTH="17%"> </TD>
<TD WIDTH="17%"> </TD>
<TD WIDTH="17%"> </TD>
</TR>

</TABLE>

</BODY>
</HTML>
```

doc_impl.tmpl

```
<TR>
<TD COLSPAN="2"><P>
<IMG SRC="/plweb/icons/sp_1_56.gif" WIDTH="400" HEIGHT="12" NATURALSIZE-
FLAG="0" ALIGN="BOTTOM"><br>
```

```
<!--The following six lines create menu items from which users can opt to
return to the search screen, return to the hitlist, see the next or previous
documents in the hit list, or jump to the first hit word in the current docu-
ment. Refer to the list of macros on page 186 for usage information for these
macros.-->

| {$TOSEARCHFORM: target=search.tmpl}Search Screen{$EOL}
| {$TOHITLIST: target=hitlist.tmpl}Results Screen{$EOL}
| {$PREVHIT: target=predoc.tmpl}Previous{$EOL}
| {$NEXTHIT: target=predoc.tmpl}Next{$EOL}
| {$TOFIRSTHIT}First Hit Word{$EOL}
| <br>

<IMG SRC="/plweb/icons/fillline.gif" WIDTH="440" HEIGHT="1" BORDER="0"
    ALT="-----"NATURALSIZEFLAG="0" ALIGN="BOTTOM">

<IMG SRC="/plweb/icons/sp_1_56.gif" WIDTH="400" HEIGHT="16" NATURALSIZE-
FLAG="0" ALIGN="BOTTOM"><br>

</TD>
</TR>
<TR>
<TD COLSPAN=3>

<b>This document, ranked number <i>{$RANK}</i> in the hitlist, was retrieved
from the <i>{$DBNAME}</i> database.</b> <p>

{$DOC: BeginTag=<FONT COLOR=ff0000><B>, EndTag=</B></FONT>}

{$IF: variable=error, operator=EQ, value=TRUE}
    {$INCLUDE: file=error_impl.tmpl}
{$ENDIF}

</TD>
</TR>
```

error.tmpl

```
<!--The error.tmpl is included by PLWeb Turbo for all errors. Typically, this
screen will not display because the first line in this template says to only
display this template if PLWeb Turbo is not in the process of displaying
another template when the error occurred. The error.tmpl will only be displayed
if no other template has been displayed yet. This will most likely only happen
during initial template implementation (e.g., if you make a request and forget
to specify the target template).-->

{$IF: variable=inline, operator=EQ, value=FALSE}

<HTML>
```



```
<HEAD>
    <TITLE>PLWeb Query Results</TITLE>
</HEAD>

<BODY BGCOLOR="#ffffff" BACKGROUND="/plweb/icons/subback.jpg" TEXT="#001572"
LINK="#008800" ALINK="#000000" VLINK="#bb4400">

<A NAME="top"></A>

<TABLE BORDER="0" CELLSPACING="0" CELLPADDING="0" WIDTH="600">

<TR>
{$INCLUDE: file=logo.tmpl}

<TD VALIGN="TOP">
    <IMG SRC="/plweb/icons/error_word.gif" WIDTH="267" HEIGHT="48" BORDER="0"
        ALT="Error" NATURALSIZEFLAG="0" ALIGN="BOTTOM"></TD>
<TD VALIGN="TOP">
    <IMG SRC="/plweb/icons/fillline.gif" WIDTH="173" HEIGHT="48" BORDER="0"
        ALT="-----" NATURALSIZEFLAG="0" ALIGN="BOTTOM"></TD>
<TD VALIGN=TOP ROWSPAN="2"><IMG SRC="/plweb/icons/error_icon.gif" HEIGHT=104
WIDTH=102></TD>
</TR>

<TR>
<TD COLSPAN="2"><P>
<IMG SRC="/plweb/icons/sp_1_56.gif" WIDTH="400" HEIGHT="16" NATURALSIZE-
FLAG="0" ALIGN="BOTTOM"><br>

<!--The following line includes the error implementation template,
error_impl.tmpl.-->

{$INCLUDE: file=error_impl.tmpl}

</TD>
</TR>

<TR>
<TD WIDTH="17%"> </TD>
<TD WIDTH="17%"> </TD>
<TD WIDTH="17%"> </TD>
<TD WIDTH="17%"> </TD>
</TR>

</TABLE>

</BODY>
</HTML>
{$ENDIF}
```

error_impl.tmpl

```
<!--This is the template included by most other PLWeb Turbo templates. It will
display error information when an error occurs while another template is being
processed by PLWeb Turbo.-->

<BR>

An error was encountered processing your request.
<P>

<!--The following line will print the identification number of the error.-->

<H3>Error <i>{$ERRORID}</i>:

<!--The following line will print the appropriate error message.-->

<FONT COLOR="0bb0b0">{$ERRORMSG}</FONT></H3> <BR>
```


C. List of Macros

Macro	Description	Page
BASEURL	Writes an HTML base HREF statement to the output for the current document.	192
CGIVAR	Writes the value of a CGI environment variable to the output.	192
CHECKED	Included for compatibility with PLWeb Turbo 2.6.	37
CPLQUERY	Writes the query that is being passed to CPL to the output.	193
CPRICE	Writes the price of a document to the output, where the price has been determined per database. Custom macro that must be used as it is shipped.	193
CONTAINER	Writes the path to the current document's container to the output.	193
CUSTOMLOGSTRING	Writes a custom log string to the output.	194
DBDESC	Writes a database description (read from the plweb.conf file) to the output.	194
DBLIST	Starts a database list loop to display the list of databases available for searching in the current view.	194
DBLIST_END	Ends the database list loop.	195
DBNAME	Writes the name of the current database to the output.	195
DICTLIST	Starts a dictionary-list loop, which is repeated over the list of dictionary advisor terms.	196
DICTLIST_END	Ends a dictionary-list loop.	196
DICTTERM	Writes the current dictionary term to the output.	196
DICTTERMID	Writes the ID of the current dictionary term to the output.	197
DOC	Writes the contents of the current document to the output.	197
DOCID	Writes the document ID of the current document to the output.	198
DOCLINK	Included for compatibility with PLWeb Turbo 2.6.	198
DOCSIZE	Writes the size of the current document (measured in characters) to the output.	199
ELSE	Starts the ELSE clause in an IF control block.	199
ENDIF	Ends an IF control block.	199
EOL	Writes an HTML link terminator () to the output if inside a link.	200
ERRORID	Writes the ID of the current error to the output.	200
ERRORMSG	Writes the message associated with the current error to the output.	200
FIRSTHITRANK	Writes the rank of the first-ranked document in the current hitlist to the output.	201
FUZZYLIST	Starts a fuzzy-list loop, which is repeated over the list of fuzzy advisor terms.	201
FUZZYLIST_END	Ends a fuzzy-list loop.	201
FUZZYTERM	Writes the current fuzzy term to the output.	201
FUZZYTERMID	Writes the ID of the current fuzzy term to the output.	202
HIDDENVALS	Writes to an HTML form a set of hidden values that allow the current user settings to be preserved.	203
HITLIST	Starts a hitlist loop.	203
HITLIST_END	Ends a hitlist loop.	204
HITSFOUND	Writes to the output the number of items found as the result of the last search.	204

Macro	Description	Page
SRCLINK	Writes to the output an HTML link that allows the user to jump to the source of the current document.	222
SUMMARY	Writes a summary of the current hit record to the output.	223
TERM	Included for compatibility with PLWeb Turbo 2.6.	37
TERMID	Included for compatibility with PLWeb Turbo 2.6.	37
TIMESTAMP	Writes a local timestamp to the output.	223
TITLE	Writes the title of the current document to the output.	224
TOFIRSTHIT	Writes to the output an HTML link that allows the user to jump to the first hit in the current document.	225
TOHITLIST	Writes to the output an HTML link that allows the user to jump to the current hitlist.	226
TOSEARCHFORM	Writes to the output an HTML link that allows the user to jump to the current search.	226
VIEW	Included for compatibility with PLWeb Turbo 2.6.	37
VIEWDESC	Writes the description associated with the current view in the `view.conf` file to the output.	227
VIEWID	Writes the ID associated with the current view to the output.	227
VIEWLIST	Starts a view-list loop, which iterates over a list of views available to the current user.	227
VIEWLIST_END	Ends a view-list loop.	228
VIEWNAME	Writes the name of the current view to the output.	228

The PLWeb Turbo suite of macros are global; that is, they are available from all templates (keeping in mind that not all macros make sense everywhere). In addition, the general format of supplying arguments to macros has changed from previous versions of PLWeb Turbo, where arguments to macros were identified by position. Macros are now identified by name, making it possible to supply only those arguments for which the default values should be overridden.

Caution

Macros inside HTML comments will be expanded just as if they were not located inside an HTML comment.

BASEURL

Writes an HTML `base HREF` statement to the output for the current document. Including this macro in HTML documents and setting the proper mapping in `baseurl.map` allows relative links to work properly. This macro should not be used if the source documents already contain a `base HREF`. An alternative method for handling relative links in HTML documents is to use the `ModifyRelativeLinks` setting in the `view.conf` file, as discussed on page 83. In that case, the `BASEURL` macro is not used.

Format

```
{$BASEURL: remoteonly=<flag>}
```

where

- `<flag>`, if `TRUE`, specifies that the macro should be expanded only for remote documents.

CGIVAR

Writes the value of a CGI environment variable to the output. Any CGI variable set by the HTTP daemon can be accessed.

Format

```
{$CGIVAR: name=<var_name>, size=<output_size>}
```

where

- `<var_name>` is the name of the CGI environment variable being accessed.
- `<output_size>` is the number of characters to use for display. If the size parameter is omitted or equals 0, the macro will use the exact number of characters needed.

Example

This following example would write the IP address of the client's machine (where the browser resides) to the output.

Request from {$CGIVAR: name=<REMOTE_HOST>}

CONTAINER

Writes the path of the current document's container to the output. If the document is a local document, the file path will be written. If the document is a remote document, the URL will be written.

Format

{$CONTAINER}

✉✉ **Note** ✉✉

If you want the user to be able to display the original document by making the container name an HTML link, you should use this macro together with the SRCLINK macro. Make sure your plweb.conf configuration file (refer to page 76) and baseurl.map file (refer to page 118) are properly set up.

CPLQUERY

Writes the query that is being passed to the CPL search engine to the output. This combines the different query components and the query rule. This macro is especially useful for testing and diagnostics when developing the query rule. For more information on the query rule, refer to "Creating the Query Rule" on page 141.

Format

{$CPLQUERY}

CPRICE

Writes the price of a document to the output, where the price is a database-wide setting determined by the value in the price.conf file. If you use this macro, you must use it as it is shipped, because it is a custom macro specifically designed to be used either as is or as an example when you are creating your own custom billing application. Any values specified in the PRICE field of documents (which is used in conjunction with the PRICE macro) will override the value set for CPRICE.

Format

{$CPRICE}

CUSTOMLOGSTRING

Writes a custom log string to the output. PLWeb Turbo allows administrators to add a `customlogstring` value to any PLWeb Turbo form. This macro can be added as a hidden value, or it can be set depending on a user setting. For example, the administrator may create several billing codes from which the end user can choose:

```
<SELECT name=customlogstring>
    <OPTION value=admin>Admin
    <OPTION value=development>Development
    <OPTION value=support>Support
</SELECT>
```

This macro writes the value of that string to the output.

Format

```
{$CUSTOMLOGSTRING size=<output_size>}
```

where

- `<output_size>` is the number of characters to use for display. If the size parameter is omitted or equals 0, the macro will use the exact number of characters needed.

DBDESC

Writes the database description assigned in the `plweb.conf` file (refer to page 77) for each database. Most commonly used within the `DBLIST` macro.

Format

```
{$DBDESC}
```

Example

```
{$DBLIST}
<input type=checkbox
{$IF: variable=SELECTEDDBS, operator=CONTAINS, value=$DBNAME}
checked
{$ENDIF}
name=dbname value="{$DBNAME}">
{$DBDESC}
{$DBLIST_END}
```

DBLIST

Starts a database list loop to display the list of databases available for the current view. This macro itself does not expand and must be terminated with the `DBLIST_END` macro. Anything between the `DBLIST` and `DBLIST_END` macros is repeated for every database listed in the `view.conf` file.

Format
```
{$DBLIST}
```

Example 1
The following example shows how to create a list of databases preceded by check boxes.

```
{$DBLIST}
<input type=checkbox name=dbname value="{$DBNAME}">{$DBDESC}
{$DBLIST_END}
```

Example 2
The following example shows how to create a drop-down menu of databases in the search screen.

```
<select name=dbname multiple>
    {$DBLIST}
    <option {$IF: variable=SELECTEDDBS, operator=CONTAINS,
    value={$DBNAME}selected{$ENDIF}
    value="{$DBNAME}">{$DBNAME}
    {$DBLIST_END}
</select>
```

The IF macro in the above example preserves your user's selection when he/she returns to the search screen. For more information on the IF macro, refer to page 203.

DBLIST_END

Ends a DBLIST loop. This macro itself does not expand and must be used with the DBLIST macro.

Format
```
{$DBLIST_END}
```

Example
See DBLIST for an example.

DBNAME

Writes the name of the current database to the output. This macro is typically used as follows:

- Inside a DBLIST loop to display the names of available databases.
- Inside a HITLIST loop (page 201) to display the name of the database for the current hit record.
- Inside a document display, to show the name of the database from which the current document was retrieved.

Format
```
{$DBNAME size=<output_size>}
```

where

- <output_size> is the number of characters to use for display. If the size parameter is omitted or equals 0, the macro will use the exact number of characters needed.

DICTLIST

Starts a loop to display the list of dictionary words. This macro itself does not expand and must be terminated with the DICLIST_END macro. Anything between the DICLIST and DICLIST_END macros is repeated for every word listed in the database dictionary.

```
{$DICTLIST}
```

Example
```
{$DICTLIST}
...
{$DICTLIST_END}
```

DICTLIST_END

Ends the loop that displays the list of dictionary words. This macro itself does not expand and must be used with the DICTLIST macro.

```
{$DICTLIST_END}
```

Example
```
{$DICTLIST}
...
{$DICTLIST_END}
```

DICTTERM

Writes the current dictionary term to the output. Most useful in the DICTLIST loop.

Format
```
{$DICTTERM}
```

Example
```
{$DICTLIST}
<option value={$DICTTERM}> {$DICTTERMID} {$DICTTERM}
{$DICTLIST_END}
```

DICTTERMID

Writes the current dictionary term ID to the output. Most useful in the DICTLIST loop.

Format

```
{$DICTTERMID: size=<output_size>}
```

where

- <output_size> is the number of characters to use for display. If the size parameter is omitted or equals 0, the macro will use the exact number of characters needed. If the ID requires less than the specified number of characters, extra spaces will be inserted, and the value will be right aligned. If the ID requires more than the specified number of characters, it will print all characters and exceed the specified length.

Example

```
{$DICTLIST}
<option value={$DICTTERM}> {$DICTERMID} {$DICTTERM}
{$DICTLIST_END}
```

DOC

Writes the contents of the current document to the output.

If the current document is an HTML document and all fields are displayed, the DOC macro expands to the contents of the <BODY>...</BODY> section of the document only, even if no explicit <BODY>...</BODY> HTML tags are present in the document. The <BODY> and </BODY> tags themselves are not included. The HTML <HEAD>...</HEAD> section is retrieved separately with the HEAD macro, which is discussed on page 200.

Format

```
{$DOC: BeginTag=<begin_tag>, EndTag=<end_tag>, field=<field>,
highlight=<highlight_flag>, format=<format_flag>, default=<default_text>}
```

where

- <begin_tag> corresponds to the begin tag for hit highlighting. The default is .
- <end_tag> corresponds to the end tag for hit highlighting. The default is .
- <field> corresponds to the field to which you wish to restrict display. The default is all fields, which means, if you do not specify a field, the content of all fields will be displayed.
- <highlight_flag> is either on or off, depending on whether you want hit highlighting to be on. The default is on.
- <format_flag> is either on or off, depending on whether you want formatting to be on. The default is on.

- <default_text> is the default text to display if no other displayable content is found. If you do not specify default text and no displayable content is found, nothing will be displayed.

Example
This will render hit words bold, perform document formatting, and display text from the text field if content is found and No Content Available if no content is found.

```
{$DOC: BeginTag=<b>, EndTag=</b>, field=text, default=No Content Available}
```

DOCID

Writes the ID of the current document in the database to the output.

Format
```
{$DOCID: size=<output_size>}
```

where

- <output_size> is the number of characters to use for display. If the size parameter is omitted or equals 0, the macro will use the exact number of characters needed. If the ID requires less than the specified number of characters, extra spaces will be inserted, and the value will be right aligned. If the ID requires more than the specified number of characters, it will print all characters and exceed the specified length.

Example
This example will write the ID of the document to the output, right aligned, using four character positions.

```
{$DOCID: size=4}
```

DOCLINK

Writes an HTML link to the output that gives the user access to the current document. The link must be terminated with an EOL macro and is used most often within the HITLIST (page 201) loop macro.

Format
```
{$DOCLINK: hrefonly=<href_flag>, target=<TargetTemplate>}
```

where

- <href_flag> is one of TRUE or FALSE. If the link should expand only to the HREF part of the link (i.e., the macro should not write the entire <A HREF=xyz> link, but only the xyz portion of the link), this value should be set to TRUE. If the entire link should be written, this value should be set to FALSE (the default value).

- <TargetTemplate> is the name of the template to which to proceed when the link is executed. For security reasons, the <TargetTemplate> cannot contain path information.

Example
The following will display the title of documents in the hitlist as links. When the user clicks on the link, PLWeb Turbo will proceed with the `predoc.tmpl` template.

```
{HITLIST}
{$DOCLINK: target=predoc.tmpl}{TITLE}{$EOL}
{HITLIST_END}
```

DOCSIZE

Writes the size of the current document (measured in characters) to the output.

Format
```
{$DOCSIZE: size=<output_size>}
```

where

- `<output_size>` is the number of characters to use for display. If the size parameter is omitted or equals 0, the macro will use the exact number of characters needed. If the integer requires less than the specified number of characters, extra spaces will be inserted, and the value will be right aligned. If the integer requires more than the specified number of characters, it will print all characters and exceed the specified length.

Example
This example will write the size of the document to the output, right aligned, using six character positions.

```
{$DOCSIZE: size=6}
```

ELSE

Starts the `ELSE` clause in an `IF` control block. This macro performs a function similar to that of `IF...ELSE` flow control structures found in many programming languages. The macro itself does not expand and is only useful with the `IF` macro. For more information, refer to the `IF` macro on page 203.

Format
```
{$ELSE}
```

ENDIF

Ends an `IF` control block. The macro itself does not expand and must be used with the `IF` macro. For more information, refer to the `IF` macro on page 203.

Format
```
{$ENDIF}
```

EOL

Writes an HTML link terminator to the output. A link terminator will be written only if PLWeb Turbo has encountered a macro that expanded to an HTML link that has not yet been terminated with the {$EOL} macro. If not, the macro does not expand. All macros that write a link to the output should be terminated with this macro.

Format
```
{$EOL}
```

ERRORID

Writes the ID of the current error to the output. This is useful mainly for support functions. The ID will be 0 if no error was encountered.

Format
```
{$ERRORID}
```

Example
```
{IF: variable=error operator=EQ value=TRUE}
Error {$ERRORID}: {ERRORMSG}
{ENDIF}
```

ERRORMSG

Writes the message associated with the current error to the output. This is useful mainly for support functions. If no error was encountered, the expansion of this macro will be empty.

Format
```
{$ERRORMSG}
```

Example
```
{IF: variable=error operator=EQ value=TRUE}
Error {$ERRORID}: {ERRORMSG}
{ENDIF}
```

FIRSTHITRANK

Writes the rank of the first-ranked document in the current section of the hitlist to the output. This is useful when allowing users to browse through sections of the hitlist, and displaying which hit records are currently being displayed.

Format
```
{$FIRSTHITRANK}
```

Example
```
Displaying documents {$FIRSTHITRANK} through {$LASTHITRANK}
```

FUZZYLIST

Starts a loop to display the list of fuzzy terms. This macro itself does not expand and must be terminated with the FUZZYLIST_END macro. Anything between the FUZZYLIST and FUZZYLIST_END macros is repeated for every word in the fuzzy list.

```
{$FUZZYLIST}
```

Example
```
{$FUZZYLIST}
...
{$FUZZYLIST_END}
```

FUZZYLIST_END

Ends the loop that displays the list of fuzzy advisor words. This macro itself does not expand and must be used with the FUZZYLIST macro.

```
{$FUZZYLIST_END}
```

Example
Refer to the FUZZYLIST macro, above.

FUZZYTERM

Writes the current fuzzy term to the output. Most useful in the FUZZYLIST loop.

Format
```
{$FUZZYTERM}
```


Example

```
{$FUZZYLIST}
<option value={$FUZZYTERM}> {$FUZZYTERMID} {$FUZZYTERM}
{$FUZZYLIST_END}
```

FUZZYTERMID

Writes the current fuzzy term ID to the output. Most useful in the FUZZYLIST loop.

Format

```
{$FUZZYTERMID: size=<output_size>}
```

where

- <output_size> is the number of characters to use for display. If the size parameter is omitted or equals 0, the macro will use the exact number of characters needed. If the ID requires less than the specified number of characters, extra spaces will be inserted, and the value will be right aligned. If the ID requires more than the specified number of characters, it will print all characters and exceed the specified length.

Example

```
{$FUZZYLIST}
<option value={$FUZZYTERM}> {$FUZZYTERMID} {$FUZZYTERM}
{$FUZZYLIST_END}
```

HEAD

Writes the HEAD portion of the current HTML document to the output. The HTML <HEAD> and </HEAD> tags themselves are not displayed. The main purpose of this macro is to allow HTML documents to be displayed using a document template, while making sure the HTML markup in the resulting document is still correct. Typically, the template would include the

```
<HTML>
<HEAD>...</HEAD>
<BODY>...</BODY>
</HTML>
```

markup, and the HEAD and BODY sections of the document would be lifted into the template with the HEAD and DOC macros. For more information on the DOC macro, refer to page 195.

Format

```
{$HEAD}
```

Example

```
<HTML>
<HEAD>
```

```
{$HEAD}
</HEAD>
<BODY>
{$DOC}
</BODY>
</HTML>
```

HIDDENVALS

Writes a set of hidden input tags inside an HTML form. This macro will preserve the following tags:

```
customlogstring
dbname
docrank
numhitsfound
numresults
operator
query
queryN
query_rule
sorting
starthit
view
```

Do not use this macro in forms where they are already defined (e.g., search screen template) but only in forms where they need to be preserved (e.g., advisor templates).

Format

```
{$HIDDENVALS}
```

Example

```
<form method=post action=fastweb>
{$HIDDENVALS}
...
</form>
```

HITLIST

Starts a hitlist loop. This macro itself does not expand and must be terminated with the HITLIST_END macro. Anything between the HITLIST and HITLIST_END macros is repeated for every hit in the hitlist being displayed.

Format

```
{$HITLIST}
```


Example
The following example would display the titles of the hit documents as hypertext links to the actual documents.

```
{$HITLIST}
{$DOCLINK: target=predoc.tmpl} {$TITLE} {$EOL}
{$HITLIST_END}
```

HITLIST_END

Ends a hitlist loop. This macro itself does not expand and must be used with the HITLIST macro. See the HITLIST macro for more information.

Format
```
{$HITLIST_END}
```

Example
Refer to the HITLIST macro, above.

HITSFOUND

Writes to the output the number of items found as the result of the last search.

Format
```
{$HITSFOUND}
```

Example
The following example would tell the user how many hits were found and how many are being displayed. The number of hits being displayed might be less than the number of hits found, depending on your setting for the number of results to display, as shown in the default search_impl.tmpl template on page 172.

```
Search found {$HITSFOUND} documents; {$HITSRETURNED} are displayed.
```

HITSRETURNED

Writes to the output the number of items returned as the result of the last search. This can be different from the number of hits found since it is limited by the number of hits requested.

Format
```
{$HITSRETURNED}
```

Example

The following example would tell the user how many hits were found and how many are being displayed. The number of hits being displayed might be less than the number of hits found, depending on your setting for the number of results to display, as shown in the default `search_impl.tmpl` template on page 172.

```
Search found {$HITSFOUND} documents; {$HITSRETURNED} are displayed.
```

IF

Starts an `IF` control block. This macro itself does not expand and must be terminated with the `ENDIF` macro. This macro performs a function similar to that of `IF...ELSE` flow control structures found in many programming languages. It takes a set of arguments describing a condition. The condition is evaluated at runtime, and if the result is `TRUE`, the template section between the `IF` and either the `ELSE` or `ENDIF` macros, whichever comes first, will be evaluated. If an `ELSE` macro is present, the section between the `ELSE` and the `ENDIF` macro is skipped. If the condition evaluates to `FALSE`, the section between the `IF` and the `ELSE` or `ENDIF` macro is skipped. If an `ELSE` macro is present, the section between the `ELSE` and the `ENDIF` macro is evaluated. `IF` control blocks may be nested.

Format

```
{$IF: variable=<variable>, operator=<operator>, value=<value>}
```

The following table explains the parameters to the `IF` macro. Most of these parameters are used in the default templates, which are discussed in detail in Appendix B. In the table below, the values for `<operator>` are as follows:

`lt`	less than
`lteq`	less than or equal
`gt`	greater than
`gteq`	greater than or equal
`eq`	equal
`ne`	not equal
`contains`	contains

Table 11-5: IF Macro Parameters

<variable>	<operator>	<value>

`Dicttermsfound`
Number of dictionary terms found
	`LT`
	`GT`
	`EQ`
	`NE`
	`LTEQ`
	`GTEQ`
		Any whole number.

`DocType`
Document type (format)
	`EQ`
	`NE`
		`CplPlsHtmlReader`
		`CplAsciiRecord`
		`CplStddoc`
		`CplAcrobatReader`
		`CPLINSOReader`

Table 11-5: IF Macro Parameters

<variable>	<operator>	<value>
Error TRUE if an error has occurred	EQ NE	TRUE FALSE
Fuzzytermsfound Number of fuzzy terms found	LT GT EQ NE LTEQ GTEQ	Any whole number.
InLine TRUE if data has already been written to the screen	EQ NE	TRUE FALSE
ItemsFound Number of hits found	LT GT EQ NE LTEQ GTEQ	Any whole number.
MaxItems Maximum number of hits to display	LT GT EQ NE LTEQ GTEQ	Any whole number.
NextNumHits Number of hits in the next section of the hitlist	LT GT EQ NE LTEQ GTEQ	Any whole number.
Nscore Normalized score (0-100)	LT GT EQ NE LTEQ GTEQ	Any number 0-100.
NumViews Number of views available to the user	LT GT EQ NE LTEQ GTEQ	Any whole number.
Operator Default search operator	EQ NE	AND OR ADJ
PrevNumHits Number of hits in the previous section of the hitlist	LT GT EQ NE LTEQ GTEQ	Any whole number.

Table 11-5: IF Macro Parameters

<variable>	<operator>	<value>
SelectedDbs List of selected databases	CONTAINS EQ NE	$DBNAME The current database. For use within an IF statement that is nested within a DBLIST loop, as described on page 192.
Sorting The order in which PLWeb Turbo should display results	EQ NE	BYRELEVANCE BYFIELD:+<field> NONE If you set <value> to BYRELEVANCE, your hitlist will be sorted by relevance. If you set <value> to NONE, for a single local database, the hitlist will be sorted in reverse document ID order. If you set <value> to BYFIELD:+<FIELD>, your hitlist will be sorted in ascending order by field. If you set <value> to BYFIELD:-<FIELD>, your hitlist will be sorted in descending order by field. Separate multiple fields by commas. For additional information on sorting, refer to "Setting Hitlist Sorting" on page 148.
Query The search query	CONTAINS EQ NE	$DICTTERM, $FUZZYTERM, or $RELATETERM The current dictionary, fuzzy, or relate term. For use within an IF statement that is nested within either a DICTLIST (page 195), FUZZYLIST (page 199), or RELATELIST (page 215) loop.
QueryType The type of operation being performed	EQ NE	simplesearch concept relate fuzzy dictionary byexample
Relatetermsfound Number of relate terms found	LT GT EQ NE LTEQ GTEQ	Any whole number.

✉✉ Note ✉✉

Variable names are not case sensitive.

Example 1

In the following example, if the number of items found is greater than 0, the {$IF}...{$ELSE} statement will be expanded. If the number of items found is less than or equal to 0, the {$ELSE}...{$ENDIF} statement will be expanded.


```
{$IF: variable=ItemsFound, operator=GT, value=0}
...
{$ENDIF}
```

Example 2

Assume the following example is an excerpt from your search screen template, where, besides enabling the user to input a query, you enable him or her to select a search operator from a drop-down control. Depending on the user's selection (and through your use of the IF macro), you load a specific template to display the user's search results.

✉✉ **Note** ✉✉

When reading this example, be sure to pay close attention to the HTML comments. They will help you to understand the code.

```
<!--This line creates the query box and passes the query via the QUERY macro to
the query rule for processing.-->
<INPUT TYPE="text" NAME="query" SIZE="45" value="{$QUERY}">

<!--This line establishes the query rule.-->
<input type = hidden name=query_rule value="($QUERY)">

<!--This block of code creates a drop-down control from which the user can
select a search operator. The search operator must be defined explicitly as
follows in order for the following IF statement to work properly. The default
operator is gotten from string 3 in defaults.cpl.-->

<b>Select a Search Operator<\b>

<select name= operator>
    <option> AND
    <option> OR
    <option> ADJ
</select>

<!--This block of code determines which template is loaded bases on the user's
operator selection.-->

{$IF: variable=operator, operator=EQ, value=adj}
    {$INCLUDE: file=adj.tmpl}
{$ELSE}{$IF: variable=operator, operator=EQ, value=and}
    {$INCLUDE: file=and.tmpl}
{$ELSE}{$IF: variable=operator, operator=EQ, value=or}
    {$INCLUDE: file=or.tmpl}
{$ELSE}
    {$INCLUDE: file=hitlist.tmpl}
{$ENDIF}
{$ENDIF}
{$ENDIF}
```

INCLUDE

Includes another template into the current template and proceeds as if the included template were part of the current template. PLWeb Turbo will resume parsing the current template after it has reached the end of the included template. Included files may be nested (e.g., an included file may include yet another file).

Format

```
{$INCLUDE: file=<file_name>}
```

where

- `<file_name>` is the name of the template file to include. For security reasons, the file name cannot contain path information. Only a file name is allowed. PLWeb Turbo will look for this file in the `<drive>:\<installDir>\plweb\views\<viewname>\templates` directory, and then in the `<drive>:\<installDir>\plweb\templates` directory.

Example

```
{$INCLUDE: file=doc_impl.tmpl}
```

For additional examples, refer to the IF macro on page 203 and to Appendix B.

INSERTAD

Writes an advertisement string to the output. When encountered, PLWeb Turbo will look for a shared library called `admngr.so` in the `<drive>:\<installDir>\plweb\lib` directory. It will then look for a C function called `getHtmlForAd()`. Any argument passed to the macro will be passed to the function. The argument will be passed literally, except that any occurrence of `"($query)"` will be replaced by the current query string. PLWeb Turbo makes no assumptions about the format of the argument. It is up to the shared library to interpret the argument.

The prototype for the `GetHtmlForAd()` function is as follows:

```
unsigned int GetHtmlforAd (
    char * s2 Args,//arguments (in)
    char * s2 Buf, //string buffer (out)
    unsigned int nBufLen);//buffer length (in)
```

Format

```
{$INSERTAD: args=<argument_string>}
```

where

- `<argument_string>` is the string to be passed to the `getHtmlForAd()` function.

Example

```
{$INSERTAD: args=www.abc.com!7568!($query)}
```


✉✉ Note ✉✉

In the example, the ! character is used to separate several sub-arguments.

LASTHITRANK

Writes the rank of last hit in the hitlist to the output.

Format

```
{$LASTHITRANK}
```

Example

```
Displaying documents {$FIRSTHITRANK} through {$LASTHITRANK}
```

MAXITEMS

Writes to the output the maximum number of items the user requested.

Format

```
{$MAXITEMS}
```

Example

```
You requested {$MAXITEMS} documents.
```

NEXTHIT

Writes to the output an HTML link that allows the user to go to the next document in the hitlist from the current document without having to return to the hitlist first. The link must be terminated with the EOL macro. This macro would normally be used on a document display screen.

Format

```
{$NEXTHIT: hrefonly=<href_flag>, target=<TargetTemplate>}
```

where

- <href_flag> is one of TRUE of FALSE. If the link should expand only to the HREF part of the link (i.e., the macro does not write the entire <A HREF=xyz> link, but only the xyz portion of the link), this value should be set to TRUE. If the entire link should be written, this value should be set to FALSE (the default value).

- <TargetTemplate> is the name of the template to which to proceed when the link is executed. For security reasons, the <TargetTemplate> cannot contain path information.

Example

```
<A HREF={$NEXTHIT: hrefonly=TRUE, target=predoc.tmpl}> Display next document.
{$EOL}
```

NEXTLIST

Writes to the output an HTML link that allows the user to go to the next section of the hitlist display. The link must be terminated with the EOL macro. This macro would normally be used on a hitlist display screen.

Format

```
{$NEXTLIST: hrefonly=<href_flag>, target=<TargetTemplate>}
```

where

- <href_flag> is one of TRUE or FALSE. If the link should expand only to the HREF part of the link (i.e., the macro does not write the entire <A HREF=xyz> link, but only the xyz portion of the link), this value should be set to TRUE. If the entire link should be written, this value should be set to FALSE (the default value).

- <TargetTemplate> is the name of the template to which to proceed when the link is executed. For security reasons, the <TargetTemplate> cannot contain path information.

Example

```
{$NEXTLIST: target=hitlist.tmpl} Display next {$NUMHITS} documents. {$EOL}
```

NEXTNUMHITS

Writes the number of hits in the next section of the hitlist to the output. This number will be equal to the maximum number of items the user requested, unless there are fewer than that number available in that section (end of the hitlist, for example).

Format

```
{$NEXTNUMHITS}
```

Example

```
{$NEXTLIST: target=hitlist.tmpl} Display next {$NUMHITS} documents. {$EOL}
```


NSCORE

Writes the normalized score for the current hit to the output. This macro would normally be used inside a HITLIST loop. The normalized score takes the absolute score returned by the engine and normalizes it to the 0 - 100 range.

Format

```
{$NSCORE: size=<output_size>}
```

where

- <output_size> is the number of characters to use for display. If the size parameter is omitted or equals 0, the macro will use the exact number of characters needed.

✉✉ **Note** ✉✉

The NSCORE macro cannot be used in the document screen template.

Example

```
{$HITLIST}
...
{$NSCORE: size=4} {$TITLE}
...
{$HITLIST_END}
```

OCCUR

For each term in the database dictionary, writes to the output the number of occurrences of the term in the dictionary. This macro would normally be used inside the DICTLIST loop, which is described on page 194.

Format

```
{$OCCUR: size=<output_size>}
```

where

- <output_size> is the number of characters to use for display. If the size parameter is omitted or equals 0, the macro will use the exact number of characters needed.

Example

```
{$DICTLIST}
...
{$OCCUR: size=6} {$TERM}
...
{$DICTLIST_END}
```

OUTPUT

Switches the destination to which the output is written. By default, anything written to the output stream is written to a symbolic destination called DISPLAY, which is the screen. The administrator can switch to a different destination in order to log information.

Format

```
{$OUTPUT: target=<target_name>}
```

where

- <target_name> is the name of the new output target. The value of this can be either DISPLAY or a symbolic name assigned to a log file. The symbolic name is associated with a log file in the view.conf file for the current view. (For more information on view.conf, refer to page 83.)

✉✉ **Note** ✉✉

Output from the remainder of the template will be sent to the new target until the output is redirected to a new target.

Example

```
...
{$OUTPUT: target=log1}
Host: {$CGIVAR: name=REMOTE_HOST}, Query: {$QUERY}
{$OUTPUT: target=DISPLAY}
...
```

PREVHIT

Writes a link to the output that allows the user to go to the previous document in the hitlist from the current document without having to return to the hitlist first. Refer to the NEXTHIT macro on page 208 for more information

Format

```
{$PREVHIT: hrefonly=<href_flag>, target=<TargetTemplate>}
```

where

- <href_flag> is one of TRUE or FALSE. If the link should expand only to the HREF part of the link (i.e., the macro does not write the entire <A HREF=xyz> link, but only the xyz portion of the link), this value should be set to TRUE. If the entire link should be written, this value should be set to FALSE (the default value).

- <TargetTemplate> is the name of the template to which to return when the link is executed. For security reasons, the <TargetTemplate> cannot contain path information.

Example
```
{$PREVHIT: target=predoc.tmpl} Display previous document {$EOL}
```

PREVLIST

Writes a link to the output that allows the user to go to the previous section of the hitlist display. This macro would normally be used on a hitlist display screen.

Format
```
{$PREVLIST: hrefonly=<href_flag>, target=<TargetTemplate>}
```

where

- <href_flag> is one of TRUE or FALSE. If the link should expand only to the HREF part of the link (i.e., the macro does not write the entire <A HREF=xyz> link, but only the xyz portion of the link), this value should be set to TRUE. If the entire link should be written, this value should be set to FALSE (the default value).

- <TargetTemplate> is the name of the template to which to return when the link is executed. For security reasons, the <TargetTemplate> cannot contain path information.

Example
```
{$PREVLIST: target=hitlist.tmpl} Display next {$PREVNUMHITS} documents {$EOL}
```

PREVNUMHITS

Writes the number of hits in the previous “section” of the hitlist to the output.

Format
```
{$PREVNUMHITS}
```

Example
```
{$PREVLIST: target=hitlist.tmpl} Display next {$PREVNUMHITS} documents {$EOL}
```

PRICE

Writes the price of the current document, as stored in the PRICE field (discussed on page 55) of that document, to the output. If the administrator has created a custom library for billing (refer to page 162 for an example), and the price of a document is not always determined solely by the content of PRICE field in the document, you should create a custom macro to replace this macro. Alternatively, you may use the sample CPRICE function.

Format

```
{$PRICE: output_size=<output_size>, denominator=<denominator>}
```

where

- <output_size> is the number of characters to use for display. If the size parameter is omitted or equals 0, the macro will use the exact number of characters needed. If the integer requires less than the specified number of characters, extra spaces will be inserted, and the value will be right aligned. If the integer requires more than the specified number of characters, it will print all characters and exceed the specified length.

- <denominator> is the value by which to divide the internal price (an integer (e.g., pennies)). The default value for the denominator is 100.

QUERY

Writes the current query to the output.

Format

```
{$QUERY: ID=<query_id>}
```

where

- <query_id> corresponds to the ID of the query in the search form. If no ID is given, the query corresponding to the query tag in the search form is written. The ID parameter is used when the search form contains fielded query tags, in which case it corresponds to the following tag in the search form:

  ```
  <input type=<a_type> name=query<query_id> ...>
  ```

Example

```
TITLE: {$QUERY: ID=1}, AUTHOR: {$QUERY: ID=2}
```

If the search form contained the following tags:

```
<input type=TEXT size=30 name=query1>
<input type=TEXT size=30 name=query2>
```

The {$QUERY: ID=1} macro would write the contents of the query1 edit control, while the {$QUERY: ID=2} macro would write the contents of the query2 edit control.

Example

```
TITLE: {$QUERY: ID=1}, AUTHOR: {$QUERY: ID=2}
```


QUERYBYEXAMPLE

Writes a link to the output that allows the user to jump to a list of documents similar to the current document.

Format

```
{$QUERYBYEXAMPLE: numterms=<num_terms>, hrefonly=<href_flag>, target=
<TargetTemplate>}
```

where

- <num_terms> is the number of terms to use in the search for similar documents (the default value is 20).
- <href_flag> is one of TRUE or FALSE. If the link should expand only to the HREF part of the link (i.e., the macro does not write the entire <A HREF=xyz> link, but only the xyz portion of the link), this value should be set to TRUE. If the entire link should be written, this value should be set to FALSE (the default value).
- <TargetTemplate> is the name of the template to which to proceed when the link is executed. For security reasons, the <TargetTemplate> cannot contain path information.

RANK

Writes the hitlist rank of the current document to the output. This macro would normally be used in hitlist display within the HITLIST loop and in document display.

Format

```
{$RANK: size=<output_size>}
```

where

- <output_size> is the number of characters to use for display. If the size parameter is omitted or equals 0, the macro will use the exact number of characters needed. If the ID requires less than the specified number of characters, extra spaces will be inserted, and the value will be right aligned. If the ID requires more than the specified number of characters, it will print all characters and exceed the specified length.

Example

```
{$HITLIST}
...
{$RANK} {$DOCLINK} {$TITLE} {$EOL}
...
{$HITLIST_END}
```

RELATELIST

Starts a relate list loop. This macro itself does not expand and must be terminated with the RELATELIST_END macro. Anything between the RELATELIST and RELATELIST_END macros is repeated for every relate term being displayed.

```
{$RELATELIST}
```

Example

```
{$RELATELIST}
...
{$RELATELIST_END}
```

RELATELIST_END

Ends the loop that displays the list of relate advisor words. This macro itself does not expand and must be used with the RELATELIST macro.

```
{$RELATELIST_END}
```

Example

```
{$RELATELIST}
...
{$RELATELIST_END}
```

RELATETERM

Writes the current relate term to the output. Most useful in the RELATELIST loop.

Format

```
{$RELATETERM}
```

Example

```
{$RELATELIST}
<option value={$RELATETERM}> {$RELATERMID} {$RELATETERM}
{$RELATELIST_END}
```

RELATETERMID

Writes the current relate term ID to the output. Most useful in the RELATELIST loop.

Format

```
{$RELATETERMID: size=<output_size>}
```


where

- <output_size> is the number of characters to use for display. If the size parameter is omitted or equals 0, the macro will use the exact number of characters needed. If the ID requires less than the specified number of characters, extra spaces will be inserted, and the value will be right aligned. If the ID requires more than the specified number of characters, it will print all characters and exceed the specified length.

Example

```
{$RELATELIST}
<option value={$RELATETERM}> {$RELATETERMID} {$RELATETERM}
{$RELATELIST_END}
```

RMTLINK

Writes to the output a link that allows the user to jump to the current remote document.

Format

```
{$RMTLINK: hrefonly=<href_flag>, target=<TargetTemplate>}
```

where

- <href_flag> is one of TRUE or FALSE. If the link should expand only to the HREF part of the link (i.e., the macro does not write the entire <A HREF=xyz> link, but only the xyz portion of the link), this value should be set to TRUE. If the entire link should be written, this value should be set to FALSE (the default value).

- <TargetTemplate> is the name of the template to which to proceed when the link is executed. For security reasons, the <TargetTemplate> cannot contain path information.

Example

```
{$RMTLINK: target=doc.tmpl} Remote document {$EOL}
```

SCORE

Writes the raw score for the current document to the output. This macro would normally be used in the hitlist display within the HITLIST loop and in document display. For more information on the HITLIST macro, refer to page 201.

Format

```
{$SCORE: size=<output_size>}
```

where

- <output_size> is the number of characters to use for display. If the size parameter is omitted or equals 0, the macro will use the exact number of characters needed. If the ID requires less than the

specified number of characters, extra spaces will be inserted, and the value will be right aligned. If the ID requires more than the specified number of characters, it will print all characters and exceed the specified length.

✉✉ **Note** ✉✉

The SCORE macro cannot be used in the document screen template.

Example

```
{$HITLIST}

...
{$SCORE}
...
{$HITLIST_END}
```

SCOREIMG

Writes an image to the output, illustrating the normalized score by means of a bar graph. The width of the component images is changed in proportion to the score.

Format

```
{$SCOREIMG: height=<graph_height>, width=<graph_width>, LeftImg=<left_img>,
RightImg=<right_img>}
```

where

- <graph_height> corresponds to the image height.
- <graph_width> corresponds to the total image width.
- <left_img> corresponds to the image to use for the left part of the graph.
- <right_img> corresponds to the image to use for the right part of the graph.

✉✉ **Note** ✉✉

The images may be in any format that is supported by the browser.

✉✉ **Note** ✉✉

The SCOREIMG macro cannot be used in the document screen template.

Example

The following example assumes that the files `rightimg.gif` and `leftimg.gif` are in the `\plweb\htdocs\icons` directory.

```
{$SCOREIMG:height=10, width=50, RightImg=\plweb\icons\rightimg.gif,
LeftImg=\plweb\icons\leftimg.gif}
```

SDETAILLIST

Starts a search details loop to display the list of highlighted entries in the document. This macro itself does not expand; rather, it creates a framework list of highlighted terms in the current document. This macro must be terminated with the `SDETAILLIST_END` macro. Anything between the `SDETAILLIST` and `SDETAILLIST_END` macros is repeated for every highlighted word in the document.

✉✉ **Note** ✉✉

The `SDETAILLIST` macro may only be used in the document screen template.

Format

```
{$SDETAILLIST}
```

Example

```
{$SDETAILLIST}
{$SDETAIL_LINK} {$SDETAIL_HIT} {$EOL} {$SDETAIL_OCCUR}<BR>
{$SDETAILLIST_END}
```

SDETAILLIST_END

Ends a search detail loop. This macro itself does not expand and must be used with the `SDETAILLIST` macro. For more information, refer to the `SDETAILLIST` macro, above.

✉✉ **Note** ✉✉

The `SDETAILLIST_END` macro may only be used in the document screen template.

Format

```
{$SDETAILLIST_END}
```

Example

```
{$SDETAILLIST}
{$SDETAIL_LINK} {$SDETAIL_HIT} {$EOL} {$SDETAIL_OCCUR}<BR>
{$SDETAILLIST_END}
```

SDETAIL_HIT

Writes the current highlighted terms to the output. This macro may be used only within the SDETAILLIST loop macro.

Format

```
{$SDETAIL_HIT: size=<output_size>}
```

where

- <output_size> is the number of characters to use for display. If the size parameter is omitted or equals 0, the macro will use the exact number of characters needed. If the ID requires less than the specified number of characters, extra spaces will be inserted, and the value will be right aligned. If the ID requires more than the specified number of characters, it will print all characters and exceed the specified length.

Example

```
{$SDETAILLIST}
{$SDETAIL_LINK} {$SDETAIL_HIT} {$EOL} {$SDETAIL_OCCUR}<BR>
{$SDETAILLIST_END}
```

SDETAIL_LINK

Writes the link to the first occurrence of the highlighted entry to the output stream. The link must be terminated with an EOL macro (discussed on page 198). This macro may be used only within the SDETAILLIST loop macro.

Format

```
{$SDETAIL_LINK}
```

Example

```
{$SDETAILLIST}
{$SDETAIL_LINK} {$SDETAIL_HIT} {$EOL} {$SDETAIL_OCCUR}<BR>
{$SDETAILLIST_END}
```

SDETAIL_OCCUR

Writes the number of occurrences for the current highlighted term to the output. This macro may be used only within the SDETAILLIST loop macro.

Format

```
{$SDETAIL_OCCUR: size=<output_size>}
```

where

- <output_size> is the number of characters to use for display. If the size parameter is omitted or equals 0, the macro will use the exact number of characters needed. If the ID requires less than the specified number of characters, extra spaces will be inserted, and the value will be right aligned. If the ID requires more than the specified number of characters, it will print all characters and exceed the specified length.

Example

```
{$SDETAILLIST}
{$SDETAIL_LINK} {$SDETAIL_HIT} {$EOL} {$SDETAIL_OCCUR}<BR>
{$SDETAILLIST_END}
```

SELECTEDDBS

Writes the list of currently selected databases to the output.

Format

```
{$SELECTEDDBS: separator=<separator_string>}
```

where

- <separator_string> is the string to use to separate the selected databases.

Example

```
{$SELECTEDDBS: separator=" "}
```

SRCLINK

Writes an HTML link to the output that allows the user to jump to the source of the current document. The document will be retrieved directly, through the HTTP daemon, without being processed by PLWeb Turbo. The link should be terminated by using the EOL macro.

✉✉ **Note** ✉✉

Not all documents can generate SRCLINK links. Only those document's whose record format is listed in the plweb.conf file will cause a link to be generated. If the record format of the current document is not listed in plweb.conf, the macro will not expand. For more information on the plweb.conf file, refer to Chapter 6.

Format
```
{$SRCLINK}
```

Example
```
{$SRCLINK} Source document {$EOL}
```

SUMMARY

Writes a summary of the current hit record to the output. This is used mainly within the HITLIST loop, or as a stand-alone summary to a single document.

✉✉ **Note** ✉✉

In the HITLIST loop, retrieving a summary is more expensive than simply retrieving the title.

Format
```
{$SUMMARY: sentences=<num_sentences>, maxchars=<max_chars>}
```

where

- <num_sentences> is the maximum number of sentences to return as the summary (the default is 3 sentences).
- <max_chars> is the maximum number of characters to use (the default is 2000 characters).

Example
```
{$HITLIST}
...
{$SUMMARY: sentences=2, maxchars=1024}
...
{$HITLIST_END}
```

TIMESTAMP

Writes a local timestamp to the output. This is often useful for logging purposes.

Format
```
{$TIMESTAMP}
```

Example
```
{$OUTPUT: target=log1}
{$TIMESTAMP}
```



```
...
{$OUTPUT: target=DISPLAY}
```

TITLE

Writes the title of the current document to the output. This macro is typically used within a HITLIST loop to customize the hitlist presented to the user. The HITLIST macro is discussed on page 201.

Format

```
{$TITLE: firstline=<first_line>, lines=<num_lines>, chars=<max_chars>,
fields=<fields>, default=<default_title>}
```

where

- <first_line> corresponds to the first line in the TITLE from the source document PLWeb Turbo should use for title display on the output. For example, if you wanted the first line of your title display to come from the fifth line of the title from your source documents, you would set this value to 5. The default value is 1.
- <num_lines> corresponds to the number of title lines to output. Using the example above, if you set firstline to 5 and lines to 3, PLWeb Turbo would display title lines 5, 6, and 7 on the output.
- <max_chars> corresponds to the maximum number of characters to output.
- <fields> corresponds to the title fields to output. When specifying multiple fields, you must separate each field with a plus sign (+).
- <default_title> corresponds to the default title to display if no other title is found. If no title is found, and you have not specified a default title, nothing will be displayed.

✉✉ **Note** ✉✉

Use of the lines and chars argument is mutually exclusive.

Example

This example would write a three-line title to the output from the AUTHOR and TEXT fields. Please note that the fields being mentioned in the fields parameter must have the TITLE attribute (T) set in the database definition file. Fields that are mentioned in this macro that do not have the TITLE attribute set will be ignored.

```
{$HITLIST}
{$TITLE: lines=3, fields=AUTHOR+TEXT}
{$HITLIST_END}
```

Example 2

The following example would display three total lines of title information, with the first line being a link to the document. If the document retrieved did not have title information, `*** UNTITLED ***` would be displayed as the title.

```
{$HITLIST}
{$DOCLINK: target=predoc.tmpl}{$TITLE: lines=1, default=*** UNTITLED
***}{$EOL}
{$TITLE: lines=2, firstline=2}
{$HITLIST_END}
```

Example 3

The following example shows the `TITLE` macro being used as a hot link in the hitlist. Assume that (1) the database in question contains a field called URL and (2) the URL field contains the actual URL to the source file.

```
{$HITLIST}
{$TITLE: fields=TITLE, lines=1}
<a href= {$TITLE: fields=URL, lines=1}>
{$TITLE: fields=TITLE, lines=1}</a>
{$HITLIST_END}
```

If the following record were to be retrieved

```
<!--plsfield:URL-->
http://www.mysite.com/

<!--plsfield:TITLE-->
Welcome to My Web Site

<!--plsfield:TEXT-->
Thanks for showing interest in my Web site...
```

The end user would see the following link to the document in the hitlist:

Welcome to My Web Site

TOFIRSTHIT

Writes an HTML link to the output that allows the user to jump to the first hit in the current document. The link should be terminated with the `EOL` macro. This is useful mainly within a document display screen and only if hit highlighting is turned on.

Format

```
{$TOFIRSTHIT: hrefonly=<href_flag>}
```

where

- <href_flag> is one of TRUE or FALSE. If the link should expand only to the HREF part of the link (i.e., the macro does not write the entire <A HREF=xyz> link, but only the xyz portion of the link), this value should be set to TRUE. If the entire link should be written, this value should be set to FALSE (the default value).

Example

```
{$TOFIRSTHIT} Jump to the first hit word {$EOL}
```

TOHITLIST

Writes an HTML link to the output that allows the user to jump back to the current hitlist. This link should be terminated with the EOL macro. This macro is normally used from within a document display screen.

Format

```
{$TOHITLIST: hrefonly=<href_flag>, target=<TargetTemplate>}
```

where

- <href_flag> is one of TRUE or FALSE. If the link should expand only to the HREF part of the link (i.e., the macro does not write the entire <A HREF=xyz> link, but only the xyz portion of the link), this value should be set to TRUE. If the entire link should be written, this value should be set to FALSE (the default value).

- <TargetTemplate> is the name of the template to which to proceed when the link is executed. For security reasons, the <TargetTemplate> cannot contain path information.

Example

```
{$TOHITLIST: target=hitlist.tmpl} Return to search results {$EOL}
```

TOSEARCHFORM

Writes an HTML link to the output that allows the user to conveniently jump to the current search screen from the search results screen or document display screen. This link should be terminated with the EOL macro.

Format

```
{$TOSEARCHFORM: hrefonly=<href_flag>, target=<TargetTemplate>}
```

where

- <href_flag> is one of TRUE or FALSE. If the link should expand only to the HREF part of the link (i.e., the macro does not write the entire <A HREF=xyz> link, but only the xyz portion of the link), this value should be set to TRUE. If the entire link should be written, this value should be set to FALSE (the default value).

- <TargetTemplate> is the name of the template to which to proceed when the link is executed. For security reasons, the TargetTemplate cannot contain path information.

Example
```
{$TOSEARCHFORM: target=search.tmpl} Return to search form {$EOL}
```

VIEWDESC

Writes the description associated with the current view in the view.conf file to the output.

Format
```
{$VIEWDESC}
```

Example
```
{$VIEWLIST}
{$VIEWID} <input type=radio name=view value={$VIEWNAME}>{$VIEWDESC}<BR>
{$VIEWLIST_END}
```

VIEWID

Writes the ID associated with the current view to the output. The view ID is simply a numerical value counting each view, starting at 1.

Format
```
{$VIEWID}
```

Example
```
{$VIEWLIST}
{$VIEWID} <input type=radio name=view value={$VIEWNAME}>{$VIEWDESC}<BR>
{$VIEWLIST_END}
```

VIEWLIST

Starts a view list loop, which iterates over a list of views available to the current user. This macro itself does not expand and must be terminated with the VIEWLIST_END macro. Anything between the VIEWLIST and VIEWLIST_END macros is repeated for every view in the list of views being displayed.

Format
```
{$VIEWLIST}
```


Example

```
{$VIEWLIST}
{$VIEWID} <input type=radio name=view value={$VIEWNAME}>{$VIEWDESC}<BR>
{$VIEWLIST_END}
```

VIEWLIST_END

Ends a view list loop. This macro itself does not expand and must be used with the VIEWLIST macro. Refer to the VIEWLIST macro for more information.

Format

```
{$VIEWLIST_END}
```

Example

```
{$VIEWLIST}
{$VIEWID} <input type=radio name=view value={$VIEWNAME}>{$VIEWDESC}<BR>
{$VIEWLIST_END}
```

VIEWNAME

Writes the name of the current view to the output.

Format

```
{$VIEWNAME}
```

Example

```
{$VIEWLIST}
{$VIEWID} <input type=radio name=view value={$VIEWNAME}>{$VIEWDESC}<BR>
{$VIEWLIST_END}
```


D. Query Operators

The following table lists and describes the PLWeb Turbo family of operators.

Table of Operators

Table 11-6: Table of Operators

Operator	Description	Example
BLANK	The default query operator will be assumed between query terms that are separated by a blank.	If OR is the default operator, **turgid concise** is interpreted as **turgid OR concise**.
OR	*Boolean OR logic operator:* Searches for records containing either of the query terms it separates. May be explicitly inserted between two query terms or set as default (i.e., implicitly inserted when no other operator is specified).	**angel OR cherub**
AND	*Boolean AND logic operator:* Searches for records containing both of the query terms it separates. May be explicitly inserted between two query terms or set as default (i.e., implicitly inserted when no other operator is specified).	**Burke AND Hare**
NOT	*Boolean NOT logic operator*: Searches for records containing the query term preceding it without containing the query term following it.	**toxin NOT arsenic**
!	*Concept operator:* For the query term preceding it, dynamically generates a list of statistically related words (within the current database) and searches for records containing those words.	If stemming is on, **run!** first generates equivalents (run, runs, running), then searches for words related to them. If stemming is off, **run!** searches only for words related to exact matches of the query term. If stemming is on, **run#!** searches only for words related to exact matches of the query term.
$	*Optional character wildcard operator:* Matches zero to one character.	**$ATF** matches *BATF* or *ATF*. **V$TOL** matches *VTOL* or *VSTOL*. **colo$r** matches *color* or *colour*.
?	*Single character wildcard operator:* Matches any single alphabetic character.	**medic???** matches *medicine, medicate.* **m?n** matches *man, men*. **??ane** matches *crane, plane, inane.*

Table 11-6: Table of Operators

Operator	Description	Example
*	*Character string wildcard operator:* Matches any string of alphabetic characters.	**medic*** matches *medics*, *medicine*, *medication*. **m*n** matches *man*, *melon*, *Manhattan*. ***ane** matches *bane*, *profane*, *insane*.
+	*Stemming operator:* Forces a search for all variants of a query term.	If stemming is off, **run+** matches *run*, *runs*, *running*.
#	*Exact match operator:* Forces a search for exact matches of a query term.	If stemming is on, **run#** matches *run*, but not *runs*, *running*.
'	*Phrase operator:* Searches for records containing exact matches of the series of query terms it encloses.	**'Project Blue Book'** is interpreted as **Project# ADJ Blue# ADJ Book#**
~	*Fuzzy search operator:* Searches for records containing words with spellings similar to any individual query term. The part of the query term that will be matched exactly is determined by the placement of the operator within a term.	**cyclo~hexene** would retrieve records containing *cyclohexene*, *cyclooctene*, and *cyclohexane*.
W/*N*	*Within operator:* Searches for records in which the query term that follows it appears within *n* words (not counting stop words) after the term preceding it.	**amphibian W/5 DNA** searches for records in which *DNA* occurs within five words after *amphibian* In order for the *WITHIN* operator to function in this manner, the database must be created with an index size of FULL or STANDARD.
ADJ , - . '	*Adjacency operator:* Searches for records in which the query term that follows it appears immediately after the term preceding it. May be explicitly inserted between two query terms or set as default (i.e., implicitly inserted when no other operator is specified). May be replaced in a query by a hyphen, apostrophe, comma, or period.	**Great ADJ white** **1,999,999** **337-81-4417** **Orcinus.orca** **O'Hara** In order for the *ADJACENCY* operator to function in this manner, the database must be created with an index size of FULL or STANDARD.
NEAR NEAR/*N*	*Near operator:* Functions as bidirectional proximity operator if a word range is specified. If no word range is specified, functions as bidirectional ADJACENCY operator.	**direct NEAR sunlight** **Nellis NEAR/10 base** In order for the *NEAR* operator to function in this manner, the database must be created with an index size of FULL or STANDARD.

Table 11-6: Table of Operators

Operator	Description	Example
SAME/*N*	*Same operator*: Searches for a query term that is within *n* paragraphs of another query term. Because *n* defaults to 0, PLWeb Turbo will automatically search for documents in which the query terms on either side of the same operator appear in the same paragraph. PLWeb Turbo recognizes two successive new line characters as a division between paragraphs. Text that does not contain a sequence of two new line characters in a row will be treated as a single paragraph.	**Africa SAME/3 Congo** would retrieve documents in which *Africa* falls within three paragraphs, before or after, *Congo* (the SAME operator is bidirectional). In order for the SAME operator to function in this manner, the database must be created with an index size of FULL. If the database is not created with an index size of FULL, the SAME operator will function like the AND operator.
NOTSAME	*Not same operator*: Searches for a query term that does not appear in the same paragraph as the second query term. Text that does not contain a sequence of two new line characters in a row will be treated as a single paragraph. Note that the NOTSAME operator is not bidirectional.	**Africa NOTSAME Congo** would retrieve documents in which *Congo* does not fall in the same paragraph as *Africa* (the NOTSAME operator is not bidirectional). In order for the NOTSAME operator to function in this manner, the database must be created with an index size of FULL. If the database is not created with an index size of FULL, the NOTSAME operator will function like the AND operator.
ATLEAST/*N*	*At least operator:* Searches for records that contain at least *n* occurrences of the query expression that immediately follows the at least operator.	**atleast/3 Clinton** would retrieve documents that contain at least 3 occurrences of the word *Clinton*. In order for the *at least* operator to function in this manner, the database must be created with an index size of FULL or STANDARD.
:	*Word-level field restriction operator*: Searches for records in which the query term preceding it appears in the field (or fieldlist or series of fields separated by commas) that follows it.	If the following statements, **tint (DST)** **tone (DST)** **fieldlist=hue tint,tone** are in the database's definition file, **blue:tint** searches for records containing *blue* in the *tint* field, and **blue:tint,tone** and **blue:hue** are both interpreted as **blue:tint OR blue:tone**

Table 11-6: Table of Operators

Operator	Description	Example
`/f:`	*Query-level field restriction operator:* Searches for records in which the query terms preceding it appear in the field (or fieldlist or series of fields separated by commas) that follows it. This does not apply to preceding query terms already assigned a field restriction operator.	If the following statements, **tint (DST)** **tone (DST)** **fieldlist=hue tint,tone** are in the database's definition file, **blue green/f:tint** searches for records containing *blue* or *green* in the *tint* field, and **blue green/f:tint,tone** and **blue green/f:hue** are both interpreted as **blue:tint OR blue:tone OR green:tint OR green:tone**
`@`	*Thesaurus operator:* Replaces the query term preceding it with synonyms from the database's thesaurus file.	If the following entry, **EBE alien xenomorph** is contained in the database's thesaurus file, **EBE@ BEM** is interpreted as **alien OR xenomorph OR BEM**
`D_`*N*	*Similar document operator:* Searches for records similar in content to the *n*th record in the database. Searches for records containing statistically significant words in the record with ID *n*. Usually retrieves record *n* itself as most relevant.	**D_69** searches for records similar in content to the 69th record in the database.
`( )`	*Scope of operation delimiters:* Function as in standard Boolean or other logic. May be nested; any level of nesting is supported. Only field restriction operators distribute through these operators.	**Dr ADJ (Howard Fine)** is interpreted as **(Dr ADJ Howard) OR (Dr ADJ Fine)** **(Arkham House):pub** is interpreted as **Arkham:pub OR House:pub**
`=`	*Equivalence operator:* Searches for records containing the specified alphanumeric value in the specified field (or fieldlist or series of fields separated by commas). Specified numeric values must be whole numbers.	**author=Suess** and **Suess=author** both search for records containing *Suess* in the *author* field, and **year=1897** and **1897=year** both search for records containing *1897* in the *year* field.

`>`
`<`
`>=`
`<=`
	Range operator:
	Searches for records in which the specified field contains an alphanumeric value in the specified range. Specified numeric values must be whole numbers. The specified field can be bounded by two specified values; only in this case may a value precede the operator.
		population>75000
		searches for records containing values greater than *75,000* in the *population* field
		Karloff<name<Pratt
		searches for records containing values alphabetically between *Karloff* and *Pratt* in the *name* field
		1951<=year<=1962
		searches for records containing values from *1951* to *1962* in the *year* field.

E. Customizing Retrieval

This chapter discusses those PLWeb Turbo system settings that you can modify to enhance searching activities and display characteristics. If you make changes to system settings, you must recreate the databases affected as well as stop and restart the daemon as follows:

1. From the START button, select Control Panel.

2. In the Control Panel window, double click the SERVICES icon.

3. In the Services window, select the `plwebd` service and click STOP and then, click START.

   After the server opens the local databases (you will get notification messages of this event), PLWeb Turbo will respond to user requests.

For more information on creating a database, refer to “Creating a Database” on page 86. In addition, if you make any changes that alter how users interact with PLWeb Turbo's search interface, remember to add an appropriate explanation to the existing online help topics.

Setting the Default Search Operator

Four different settings determine which search operator PLWeb Turbo will use when executing searches. It is recommended, however, that you define your operators in the search form and in the query rule rather than in the `defaults.cpl` and `view.conf` files.

1. String 3 in `defaults.cpl`, which is used only if the operator setting is undefined by the administrator or left blank by the end user. For more information, refer to “Setting the Default Search Operator” on page 237.

2. The search operators defined explicitly in the query rule, which override the setting for `Default-Operator`, as explained below. For more information, refer to “Creating the Query Rule” on page 133.

3. The `DefaultOperator` key in the `view.conf` file, which determines the search operator used to connect components in the query rule, in the event that no operator is explicitly stated in the query rule. For more information, refer to “Configuring the view.conf File” on page 80.

4. The `operator` setting in the search form, which determines which search operator will be used for the entire query (or within fields) This setting overrides the setting in the `defaults.cpl` file. For more information, refer to “search_impl.tmpl” on page 159.

Editing defaults.cpl

You can alter certain characteristics of PLWeb Turbo by changing settings in `defaults.cpl`, an ASCII text file located in the <drive>:`\<installDir>\plweb\cpl` directory that contains numbered strings of text, each of which dictates a database definition or software configuration default for the PLWeb Turbo installation site. While certain settings can be superseded at the database level, you can modify a few site-wide defaults by changing the corresponding lines in `defaults.cpl`.

For more detailed descriptions of any searching operations, please consult the Online Searching Help, which is included with your PLWeb Turbo distribution.

Caution

You should change *only* those `defaults.cpl` strings that are discussed in this document. Furthermore, when editing a given string, you may change only its setting; the string number must remain intact and there *must not* be a trailing space at the end of the string (i.e., a carriage return must immediately follow the value). If the options for a string are yes or no, the valid entries are capital `Y` or `N` only.

Setting the Default Search Operator

`Defaults.cpl` string 3 defines the default operator for PLWeb Turbo searching; the default value is `OR`. The valid values for this string are `OR`, `AND`, and `ADJ`.

Setting the Default for Search Stemming

`Defaults.cpl` string 287 determines the default status for search stemming in PLWeb Turbo. By default, it is turned on. If you want search stemming turned off by default, you can change string 287 to be `N`.

Note

If you turn search stemming off in this way, you should add to the existing online help topics an explanation of the stemming operator, similar to the following example.

Finding Word Variants

Instead of searching only for exact matches of your query terms when search stemming is off, you can search for word variants by using the *stemming operator*. You append this operator, the plus sign, to the end of the query term for which you wish to find variants.

Query Format

```
<word>+
```

Example

```
runs+
```

This query would search for alternate forms of the word *runs*, including *run*, *runner*, *running*, etc. When stemming is on, the stemming operator is not needed, because the default behavior is to find word variants.

Modifying Fuzzy Matching Selectivity

The selectivity of PLWeb Turbo's fuzzy matching algorithm is determined by `defaults.cpl` string 138, for which the default value is 60. You can increase or decrease fuzzy matching selectivity by respectively raising or lowering the string setting. The valid values for this string are 1 through 100.

✉✉ Note ✉✉

Slight adjustments to this string setting can result in wide variations in fuzzy matching selectivity.

Activating Automatic Thesaurus Operation for All Query Terms

By modifying string 227, you can arrange for PLWeb Turbo to implicitly add the thesaurus operator, *@*, to all words in a query. By default, this function is disabled; to activate automatic thesaurus operation on all query words, you can change string 227 to be Y.

With this function enabled, the following query

```
lions AND tigers AND bears
```

would be interpreted by PLWeb Turbo as

```
lions@ AND tigers@ AND bears@
```

Setting the Maximum Number of Lines Scanned for the Hitlist Display Attribute

You can set the maximum number of lines within a record PLWeb Turbo will scan for fields with the `T` attribute with `defaults.cpl` string 238. The default value for this string is 50; even if a field is defined for hitlist display, its contents will not appear in the hitlist unless it occurs within the first 50 lines of its parent record. For more information on the `T` attribute, refer to Table 5-1 on page 58.

Modifying the Stopword List

As opposed to a keyword-based system, PLWeb Turbo is full-text retrieval software: it indexes every word in a document, with the exception of *stopwords*. Stopwords are those terms that PLWeb Turbo is programmed to ignore during the indexing and retrieval processes, in order to prevent the retrieval of extraneous records. Generally, a stopword list includes those articles, pronouns, adjectives, adverbs, and prepositions (*the, they, very, not, of,* etc.) that are most common. For a complete list of default stopwords, refer to page 239.

You can arrange for specific words to be ignored during searching and indexing by adding them to the stopword list that is contained in the ASCII text file `stopword.cpl`, located in `<drive>:\<installDir>\plweb\cpl\latin1\english`. Conversely, you can have existing stopwords recognized during indexing and searching by deleting them from this file.

💣💣 Caution 💣💣

Because all existing databases must be re-indexed when this stopword list is altered, you should make all global stopword modifications when PLWeb Turbo is first installed.

When editing `stopword.cpl`, you should observe the following guidelines:

- Words must be separated by a new line.
- If changes to the stopword list are to apply to an existing database, the database must be rebuilt after the stopword list has been altered.

✉✉ **Note** ✉✉

When considering stopwords, you should bear in mind that any words listed in `stopword.cpl`

Default Stopword List

A
a
about
above
across
adj
after
again
against
all
almost
alone
along
also
although
always
am
among
an
and
another
any
anybody
anyone
anything
anywhere
apart
are
around
as
aside
at
away

B
be
because
been
before
behind
being
below
besides
between
beyond
both
but
by

C
can
cannot
could

D
deep
did
do
does
doing
done
down
downwards
during

E
each
either
else
enough
etc
even
ever
every
everybody
everyone
except

F
far
few
for
forth
from

G
get
gets
got

H
had
hardly
has
have
having
her
here
herself
him
himself
his
how
however

I
i
if
in
indeed
instead
into
inward
is
it
its
itself

J
just

K
kept

M
many
maybe
might
mine
more
most
mostly
much
must
myself

N
near
neither
next
no
nobody
none
nor
not
nothing
nowhere

O
of
off
often
on
only
onto
or
other
others
ought
our
ours
out
outside
over
own

P
p
per
please
plus
pp

Q
quite

R
rather
really

S
said
seem
self
selves
several
shall
she
should

since
so
some
somebody
somewhat
still
such

T
than
that
the
their
theirs
them
themselves
then
there
therefore
these
they
this
thorough
thoroughly
those
through
thus
to
together
too
toward
towards

U
under
until
up
upon

V
v
very

W
was
well
were
what
whatever
when
whenever
where
whether
which
while
who
whom
whose
will
with
within
without
would

Y
yet
young
your
yourself

F. Command-Line Invocation

Why Execute fastweb from the Command Line?

Administrators of PLWeb Turbo have doubtless come up with a multitude of uses for the `fastweb` executable, from automating repetitive tasks, to trouble shooting, to customizing searching as the end user experiences it. This guide could not begin to address the countless uses for `fastweb`; therefore, we have narrowed the topic down to two of the most frequent implementations: automating administrative tasks and troubleshooting.

Automating Administrative Tasks

Assume, for example, that you need to delete documents from a database on a daily basis, and you do not want to do this manually. Now assume that

- Documents in your database have a non-displaying field that contains the date the documents should be deleted from the database.
- You have a view set up for administrative tasks.
- You have a hitlist template in the administrative view configured such that it returns document IDs only.

Using the above assumptions, you could automate document deletion by running `fastweb` from the command line, specifying

- The administrative view.
- The hitlist template that returns docids only.
- A field-restricted search on the non-displaying field for today's deletion date.

You could then pass the returned docids to `pldelete`.

Troubleshooting

In anticipation of having to perform troubleshooting from the command line, set up a view specifically for troubleshooting. The troubleshooting view should use the default templates and configuration settings so you can add in changes one at a time until you find the problem. Having this view will keep you from affecting other templates and configuration settings you have established for production.

Assume for a moment that your PLWeb Turbo application is getting an error in the Web browser, but you cannot tell if the error is coming from the HTTP server or from PLWeb Turbo. Running `fastweb` from the command line can help determine what PLWeb Turbo is doing and in the process, help identify the location of the problem.

Setting Up the Environment

For command-line invocation of `fastweb.exe` to work, you must:

1. Make sure the environment variables listed under “Checking the Environment Variables” on page 42 are set.

2. Set the following environment variables. Arguments to the QUERY_STRING environment variable are discussed under “Setting the QUERY_STRING Environment Variable” on page 245.

```
set REQUEST_METHOD GET
set QUERY_STRING "<key1>=<val1>&<key2>=<val2>&<key3>=<val3>..."
```

Setting the QUERY_STRING Environment Variable

Once you define it, the QUERY_STRING variable holds the default values that will get executed when you invoke the fastweb executable. If you do not specify a value for a particular key, the default value (listed in the table below) will be used.

Table 11-7: Keywords and Values: QUERY_STRING

Key	Description	Default Value
customlogstring	Specifies a custom log string to add to the log.conf file.	""
dbname	Specifies the name of the database to search. The database name you specify must correspond to a name listed in the plweb.conf file.	NONE
docdb	Specifies the database from which to retrieve a document	NONE
docid	Specifies the document ID in the database.	0
numresults	Specifies the number of results to return.	25
operator	Specifies the default search operator.	Determined by string 3 in defaults.cpl or by the operator specified in view.conf for the view you are searching.
query	Specifies your query string. Words in the string must be separated by the plus sign.	""
queryN	Specifies a fielded query string. The field must be separated from the query by a colon. Words in the query must be separated by the plus sign.	""
setCookie	Specifies whether you want an HTTP cookie.	0
sorting	Specifies the sorting mode. Options are byrelevance none byfield[+|-]:def	byrelevance
TemplateName	Specifies which template to return. This is a mandatory setting.	NONE
view	Specifies the name of the view to search. The view name you enter must correspond to a view name listed in a view.conf file.	NONE

QUERY_STRING Examples

The following example settings for QUERY_STRING assume the default shipping templates.

Return the View Selection Screen or the Search Screen if there is only one view.
`"TemplateName=views.tmpl"`

Return the Search Screen.
`"view=view1&TemplateName=search.tmpl"`

Perform a search for *Canada* against the TRAVEL database and return a hitlist.
`"view=view1&TemplateName=prehit.tmpl&dbname=Travel&query=canada"`

Retrieve document ID 508 from the TRAVEL database and perform highlighting for the *Canada* query.
`"view=view1&query=canada&docid=508&docdb=Travel&TemplateName=predoc.tmpl"`

✉✉ **Note** ✉✉

While you are customizing your templates, you may want to consider turning off preparsing of templates by adding the key and value `PreparseTemplates=False` to the `view.conf` file (refer to Table 7-2 on page 83). Turning off preparsing of templates enables you to modify and debug templates without having to restart the daemon for changes to those templates to take affect (i.e., any changes you make will be effective immediately). When you have finished customizing your templates, turn preparsing of templates back on by removing the `PreparseTemplates` key and value from the `view.conf` file.

Running the fastweb Executable

Once you have your `QUERY_STRING` variable set, you are ready to execute the `fastweb.exe` command. The `fastweb.exe` command must be executed from the `<drive>:\<installDir>\cgi-bin` directory. Alternately, you can set the `PLWEB_ROOT` environment variable to the `<drive>:\<installDir>\plweb` directory and execute it from any directory.

Using fastweb to Ping the PLWeb Daemon

You can ping the PLWeb daemon (`plwebd`) by executing `fastweb.exe -p` from the command line. If the daemon is alive, it will respond by returning its build date and time, as follows:

```
Feb  4 1998 08:47:56
```

If the daemon is not alive (i.e., you fail to connect), you will receive a short HTML page indicating the failure to connect. If the daemon is hung, `fastweb` will continue to wait for `plwebd`, without producing any output.

Index

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W