



***WebSPIRS™***  
***Implementor's Guide***

Document Version 3.0  
WebSPIRS Version 3.0

# *WebSPIRS™ Implementor's Guide*

Copyright 1996, 1995 © SilverPlatter International N.V.

All rights reserved.

The software described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license. Possession, use, duplication, or dissemination of the software described in this documentation is authorized only pursuant to a valid written license from SilverPlatter Information, Inc.

SilverPlatter is a registered trademark of SilverPlatter International N.V.

Blat is in the public domain and was written by Pedro Mendes and Mark Neal at the University of Wales Aberystwyth

Linux has been placed under the GNU Public License.

Microsoft, MS-DOS, Microsoft CD-ROM Extensions, Microsoft Windows, and Windows NT are registered trademarks of Microsoft Corporation.

Netscape is a trademark of Netscape Communications Corporation.

NFS and Solaris are trademarks of Sun Microsystems, Inc.

PolyMake is a registered trademark of INTERSOLV, Inc.

SCO is a trademark of The Santa Cruz Operation, Inc.

UNIX is a registered trademark of UNIX System Laboratories, Inc.

All other names, products, and services are trademarks or registered trademarks of their respective holders.

## **SilverPlatter Information, Inc.**

---

100 River Ridge Drive

Norwood, MA 02062-5062, USA

TEL: 617-769-2599

FAX: 617-769-8763

10 Barley Mow Passage

Chiswick, London, W4 4PH, England

TEL: 081-995-8242

FAX: 081-995-5159

Email address: [staff@silverplatter.com](mailto:staff@silverplatter.com)

Internet address: <http://www.silverplatter.com>

# Table of Contents

Introduction	vii
Intended Audience	vii
Document Structure	vii
Related Documentation	viii
Conventions	viii
Reader's Comments	viii
<b>Chapter 1 - Overview</b>	<b>1-1</b>
WebSPIRS Process	1-1
The cgibaby Process	1-2
The cgichild Process	1-2
The cgiadult Process	1-2
Log Files	1-3
<b>Chapter 2 - Installing and Configuring</b>	<b>2-1</b>
Preparing to Install WebSPIRS	2-1
Installing WebSPIRS	2-1
Linux	2-2
Solaris	2-3
Windows NT	2-5
Configuring WebSPIRS	2-6
The webspirs.cfg and cgibaby.cfg Files	2-6
The erlcnt.cfg File	2-7
The .htaccess File	2-8
The mime.types File	2-8
Opening WebSPIRS On Your Browser	2-8
<b>Chapter 3 - Tutorial for Customizing Templates</b>	<b>3-1</b>
What are the WebSPIRS templates?	3-1
What do the encoded macros do?	3-1
Why Customize a Template?	3-1
How can I edit a template?	3-2
The Editable and Processed Template States	3-2
The FORM in WebSPIRS	3-4
What kinds of customizations can I make?	3-4
Bypassing the Login page	3-4
Preselecting Databases	3-5
Displaying Specific Fields	3-5
Changing the Records Display Default Number	3-6
Changing a Template Title	3-6

Changing the Graphical Interface	3-6
Creating a Table	3-7
Controlling the Flow of Pages	3-8
WebSPIRS Templates	3-9
Complete Pages	3-9
Large Fragments	3-11
Utility Fragments	3-11
<b>Chapter 4 - Creating and Using Macros</b>	<b>4-1</b>
Understanding the Macros	4-1
Nested Macros	4-2
SilverPlatter Macros	4-2
Interface-Specific Variables	4-2
Administration Macros	4-6
General Database Macros	4-7
Text Display Macros	4-9
Search Macros	4-13
Automatic Subject Lookup Macros	4-14
Field-Specific Index Macros	4-14
Field List Macro	4-16
Thesaurus Term Macros	4-16
Guide Keyword Macros	4-19
Marked Records	4-20
Miscellaneous Macro Support	4-21
Adding Macros	4-24
C++ Language Implementation	4-25
Useful Services Available to the Macro	4-28
<b>Chapter 5 - WebSPIRS Class Library</b>	<b>5-1</b>
Class cgi_Config	5-2
Public Member Functions	5-2
Class dxp_to_html	5-5
Public Member Functions	5-5
Class erlAdmin	5-6
Protected Member Functions	5-6
Class erlAdmin_DBInfo	5-7
Public Member Functions	5-7
Class erlAdmin_UserInfo	5-8
Public Member Functions	5-8
Class sgml_Field	5-10
Public Member Functions	5-10
Class www_Admin	5-11
Public Member Functions	5-11
Class www_Alert	5-12
Public Member Functions	5-12
Class www_Arguments	5-13

Public Methods	5-13
Class <code>www_Database</code>	5-15
Public Member Functions	5-15
HTML Expansion Methods	5-15
Class <code>www_Environment</code>	5-18
Public Member Functions	5-18
Class <code>www_ERLConnection</code>	5-20
Public Member Functions	5-20
Protected Member Functions	5-22
Class <code>www_Field</code>	5-24
Public Member Functions	5-24
Class <code>www_FSI</code>	5-25
Public Member Functions	5-25
Class <code>www_Guide</code>	5-27
Public Member Functions	5-27
Class <code>www_HTML_Helper</code>	5-28
Public Member Functions	5-28
Macro Expansion Methods	5-29
Utility Methods	5-29
Utility Methods for Generating HTML	5-31
Table of Helpers Methods	5-32
Protected Methods	5-33
Class <code>www_Macro</code>	5-34
Public Member Functions	5-34
Class <code>www_MacroCaller</code>	5-36
Public Member Functions	5-36
Class <code>www_Record</code>	5-37
Public Member Functions	5-37
Record Extraction Methods	5-37
Class <code>www_Request</code>	5-41
Public Member Functions	5-41
Class <code>www_Search</code>	5-44
Public Member Functions	5-44
Class <code>www_Server</code>	5-46
Public Member Functions	5-46
Class <code>www_Template</code>	5-48
Rules Used in <code>www_TemplateExpander:Lookup</code> to Turn Macros into HTML	5-48
Public Member Functions	5-48
Class <code>www_User</code>	5-49
Public Member Functions	5-49
Class <code>www_Wild</code>	5-51
Public Member Functions	5-51

<i>Chapter 6 - Frequently Asked Questions</i>	<u>6-1</u>
<i>Glossary</i>	<u>G-1</u>

# Introduction

WebSPIRS is SilverPlatter's Information Retrieval System for the World Wide Web (WWW). It is a common gateway interface (CGI) application which allows any forms-capable browser, such as Netscape, to search SilverPlatter (SP) Electronic Reference Library (ERL) databases available over the Internet.

## Intended Audience

This guide has two instructional parts and the audience for each can be different:

- **Tutorial for Customizing Templates.** Chapter 2 of the guide provides instructions for customizing WebSPIRS interface forms, which result from the processing of templates created with HyperText Markup Language (HTML). If you have access to a web browser, such as Netscape or Mosaic, you can use these instructions to customize the templates to your own specific database needs.
- **Creating and Using Macros.** Chapter 3 of the guide provides instructions for creating more macros than are provided by SilverPlatter. If you want to implement a more complex scheme for searching or if you have some other need which involves creating new SP macros, you should have knowledge of the C++ programming language. You will also need code libraries and documentation, which can be obtained by participating in SilverPlatter's WebSPIRS Developer Program. For additional information contact Yogen Pathak, Software Development Manager, by electronic mail at [YogenP@SilverPlatter.com](mailto:YogenP@SilverPlatter.com).

## Document Structure

This document contains the following:

Chapter 1, *Overview*, provides an introduction to the SilverPlatter WebSPIRS technology.

Chapter 2, *Installing and Configuring WebSPIRS*, provides instructions for installing and configuring WebSPIRS on the Linux, Solaris, and NT platforms.

Chapter 3, *Tutorial for Customizing Templates*, provides instructions and examples for customizing the WebSPIRS templates, and a description of each WebSPIRS template.

Chapter 4, *Creating and Using Macros*, provides instructions and examples for creating new SilverPlatter macros and describes each of the existing macros.

Chapter 5, *WebSPIRS Class Library*, contains a hierarchical class drawing of the WebSPIRS classes and provides detailed information for each class and its methods.

Chapter 6, *Frequently Asked Questions*, contains frequently asked questions about WebSPIRS and the answers to those questions.

The Glossary defines some of the terms used in this guide.

The Index helps you find information contained in this guide.



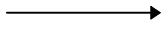
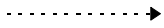
## Related Documentation

You should have access to the following documentation:

- *SilverPlatter CORE Reference Manual, Version 2.0*
- *SilverPlatter CORE Wrapper Reference Manua, Version 1.0*

## Conventions

The conventions used in this guide are described in the following table:

Convention	Description
<b>boldface text</b>	Boldface text represents WebSPIRS class names, member functions, commands, and format specifiers. Within syntax descriptions, boldface type indicates text that must be entered exactly as shown.
<i>italic text</i>	Words in italics indicate placeholders for information you must supply, such as a value for a parameter. Italics are also used for book titles.
<u>underlined text</u>	Underlined text is used for emphasis.
Monospaced text	Monospaced type indicates code examples and text as it appears in a program or on a screen. It is also used to represent file names, templates, and SilverPlatter macros; for example, <code>wwwdb.hpp</code> , <code>search.htm</code> , and <code>sp.dbid.p</code> .
	In graphical representations, an arrow with a solid line indicates direct inheritance from one class to another.
	In graphical representations, an arrow with a broken line indicates a relationship between two classes.

## Reader's Comments

We welcome and encourage comments on the usability of this manual. Please send any comments or suggestions for improvement to the following electronic mail address:

[ElizK@SilverPlatter.com](mailto:ElizK@SilverPlatter.com)

---

# Chapter 1 - Overview

---

WebSPIRS provides access to SilverPlatter ERL databases using a Web browser. It is a client that talks to a server. Forms created from templates in HTML format retrieve the ERL data using WebSPIRS. You can use the WebSPIRS templates as models to create your own interface forms, or you can make a copy of a template and modify it to fit your needs. See Chapter 3 for details.

The source templates for the WebSPIRS interface forms depend upon common gateway interface (CGI) scripts which allow web servers to interact with external processes. There is a context-specific template for each state of the user's interaction with the ERL server.

The following major features are supported by WebSPIRS:

- Login
- Database selection
- Search
- Index
- Record display
- Marked records
- Thesaurus
- Automatic subject lookup
- Database-specific help
- Page-specific help and help on search methods.

Because WebSPIRS works with the ERL Technology, it supports only the features provided by the most current version of ERL.

## WebSPIRS Process

WebSPIRS consists of three internal processes which move information from the web browser to the ERL server and back again. These three processes are:

- **cgibaby**
- **cgichild**
- **cgiadult**

## The cgibaby Process

The **cgibaby** process is launched by the HTTP server. It reads the HTTP request from its **stdin**. To this it attaches the following:

- Its environment variables. These provide it with information such as `REMOTE_HOST`, `SERVER_NAME`, and `SCRIPT_NAME`.
- Its command line arguments
- A unique pipe port in which it expects to find the completed HTML form.

The **cgibaby** process writes a request to a pipe that is connected to the **cgichild** dispatcher process. It waits for the HTML form to appear in its pipe, and it copies the form to **stdout**. It then exits and the form goes from the HTTP server to the Web browser.

## The cgichild Process

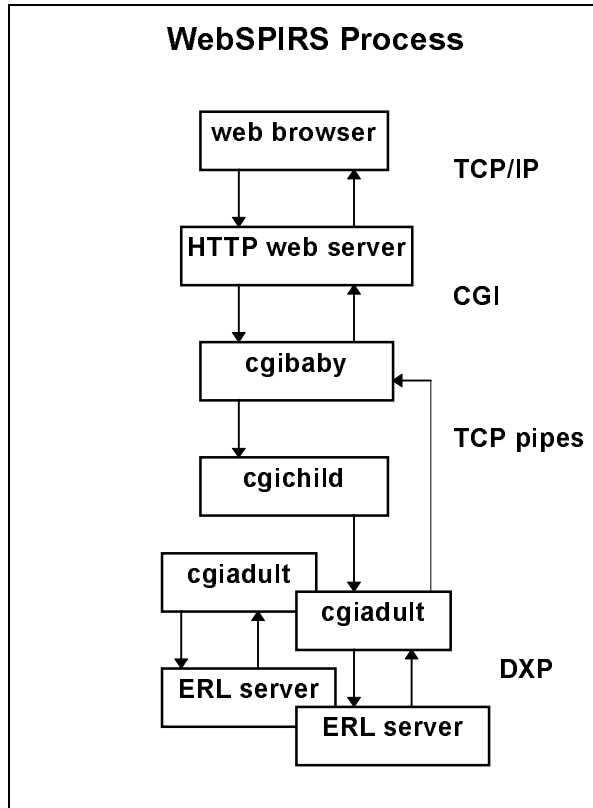
The **cgichild** process dispatches HTTP requests to the appropriate ERL connection (**cgiadult**). ERL must be running for WebSPIRS to operate. On startup, **cgichild** creates a pipe for **cgibaby** to write to and waits for requests to come in. On receiving a request it goes through the following steps:

1. It looks for a "user number" (SP macro, `sp.usernumber.p`).
2. If none is found, a new user is assumed and a new user number is assigned and attached to the request. (SP macros--`sp.username` and `sp.password`).
3. If one is found, it checks a table to see if the user's user name and password are known, and if so, they are attached to the request.
4. If the user is not currently attached to a connection (this is determined by the same table), the user is assigned the least recently used connection.
5. The request is then written to the named pipe associated with the connection.
6. **cgichild** goes back to waiting for a request.

## The cgiadult Process

The **cgiadult** process manages a connection with ERL servers. It interprets the HTTP request, loads and interprets the HTML template, builds the output HTML form, and writes the form to the output pipe assigned by **cgibaby**. It attaches its environment variables and command line arguments to the incoming request. This allows it to be used as a standalone gateway or a standalone debuggable module which is convenient for working on new macros.

The following steps show the flow of information in the WebSPIRS process:



1. The user of the Web browser fills out the form to get data.
2. The HTML request goes to the HTTP Web server, and a **cgibaby** process is launched.
3. The CGI request is read by **cgibaby**.
4. The **cgibaby** process sends the CGI request data to the **cgichild** process through a TCP pipe.
5. The **cgichild** process sends the request to **cgiadult** using the TCP pipe associated with the connection.
6. Data eXchange Protocol (DXP) messages go between a **cgiadult** process and the ERL server.
7. The HTML template stored on the Web server is filled with DXP data after **cgiadult** reads it.
8. The **cgiadult** completes the page, sends it back to **cgibaby**, and the CGI script ends. The Web server reads the result and forwards it back to the Web client where it is displayed to the user.

## Log Files

Various logs are written as the **cgibaby**, **cgichild**, and **cgiadult** processes take the information from the web browser through WebSPIRS and back again. These files will only appear if `debug=` in the `webspirs.cfg` file is set to something over than zero. The logs written are as follows:

- `wwwbaby.req` is the last request that a **cgibaby** received. This includes the add-on environment variables. It may be used to directly run **cgibaby**. In the following command, **cgibaby** reads the request from the `wwwbaby.req` file.

```
cgibaby READ_FROM=wwwbaby.req
```

Note that for this to work fully, a **cgichild** process must be up and running. Also, the request will not contain a username and password; these must be added manually.

- `webuser.log` is a log of users, IP addresses, and Host names (if available). It is written by **cgibaby**. Note: This file is not updated or cleaned automatically. You must erase information yourself.
- `wwwlast.req` is the last request received by a **cgiadult**. It includes the username and password and will function with ERL. It may be fed directly into **cgiadult** or **cgibaby**. In the following command, **cgiadult** reads the request from the `wwwlast.req` file:

```
cgiadult READ_FROM=wwwlast.req
```

- `webreq.log` is a log of all requests that have come in since the last time the file was erased. Requests can be extracted from here using an editor and then fed into **cgiadult** or **cgibaby** as above.
- `wwwform.log` is a log of all the forms written to WebSPIRS. It is useful for determining what an incomplete connection produced.

## Chapter 2 - Installing and Configuring

---

This chapter provides instructions for installing WebSPIRS on the Linux, Solaris, and NT platforms. It also includes instructions for editing several files to configure the WebSPIRS software

### Preparing to Install WebSPIRS

Before you install, you will need the following software:

- The WebSPIRS client file package

You can download the appropriate WebSPIRS package file for your platform from the SilverPlatter **ftp** site (<ftp://ftp.silverplatter.com/software/erl-clients/>). The name of the file will probably be something like `ws30b9.tgz`, but for the purposes of this document it is called simply `webspirs`; for example, `webspirs.tgz` for Linux, `webspirs.pkg` for Solaris, and `webspirs.exe` for NT.

- A Hypertext Transfer Protocol (HTTP) server

If you do not already have a HTTP server, install, configure, and start a server on your machine. You can download a free server from the Internet:

- ⇒ The Apache server (<http://www.apache.org/>) supports Linux and Solaris platforms. Look in their `binaries` directory (<http://www.apache.org/dist/binaries/>) for platform-specific files.
- ⇒ The WebSite server (<http://software.ora.com/>) from O'Reilly & Associates is a free NT Web server. Also for use with NT is a free server from EMWAC (European Microsoft Windows NT Academic Centre) at <http://emwac.ed.ac.uk/>.

- A compatible Web browser for testing

Web browsers used at SilverPlatter include Netscape and Lynx. You can download the Netscape browser from Netscape (<http://cgi.netscape.com/>).

### Installing WebSPIRS

This section provides instructions for installing WebSPIRS on the Linux, Solaris, and NT platforms. If you have a previous version of WebSPIRS installed on your system and you want to save changes you have made to templates and other files, do the following:

1. Backup anything you need to save including any configuration files (`.cfg`).
2. Remove the old files.
3. Install the new files using the instructions described in this section.
4. Integrate your saved files with the new files.

## Linux

Before you install WebSPIRS on a Linux platform, you should have the following minimum hardware requirements:

- 386DX/33 MHz IBM-compatible PC
- 16 MB RAM (1 to 10 simultaneous users). Add 1.5 MB per user over 10 users.
- 4 MB free hard disk space
- Linux Version 1.2.13+ (RedHat recommended)
- Standard video
- TCP/IP network protocol
- ERL server (separate machine)

To install WebSPIRS on Linux, complete the following steps:

1. Log in as root on your machine.
2. Change to the directory to which you downloaded the `webspirs.tgz` files.
3. If you are running a previous version of WebSPIRS, back up anything you need to save and remove the old files.
4. Either run the slackware `pkgtool` on the `webspirs.tgz` file or manually install WebSPIRS.

### Installing with the `pkgtool`

`pkgtool` modifies the `/etc/rc3.d/rc.local` file to start WebSPIRS at bootup and unpacks the WebSPIRS files into the following directory structure:

```
/usr/local/etc/webspirs/bin
/usr/local/etc/webspirs/url/doc
/usr/local/etc/webspirs/url/images
/usr/local/etc/webspirs/template
/usr/local/etc/webspirs/template/help
```

`pkgtool` also creates a symbolic link to `webspird.cgi` and `webspirs.cgi` in the `/usr/local/etc/httpd/cgi-bin` directory.

### Installing Manually

If you do not have the slackware `pkgtool`, you can manually install WebSPIRS:

```
cd /
tar -xvzf /<full_path>/webspirs.tgz
vi /install/doinst.sh
/install/doinst.sh (execute doinst.sh)
rm -r /install
```

Note that the install directory can be removed after execution. When you edit the `doinst.sh` file, edit the following lines:

- Change `WEBSPIRSUSER=` to `nobody`
- Change `WEBSPIRSGROUP=` to `nobody`
- Change `CGIBIN=` and `DOCDIR=` if your web server is in a different location.

5. As described in the “Configuring WebSPIRS” section, you need to change the WebSPIRS default configuration settings in the `webspirs.cfg` file and the ERL server address information in the `erlclnt.cfg` file.

6. Start the WebSPIRS client from an XTerm to record any debug information in the event of a problem:

```
su nobody -c "/usr/local/etc/webspirs/bin/webspird.cgi restart"
```

Or reboot your machine to start up the WebSPIRS processes automatically.

At this point, an end user can point a Web browser at the WebSPIRS client and perform searches. See the section, “Opening WebSPIRS On Your Browser.”

---

## Solaris

Before you install WebSPIRS on a Solaris platform, you should have the following minimum hardware requirements:

- Sun SPARCstation 10
- 32 MB RAM (1 to 10 simultaneous users). Add 1.5 MB per user over 10 users.
- 3 MB free hard disk space
- Sun Solaris Version 2.3+
- Standard video
- TCP/IP network protocol
- ERL server (separate machine)

To install WebSPIRS on Solaris, complete the following steps:

1. Log in as `root` on your machine.
2. Change to the directory to which you downloaded the `webspirs.pkg` files.
3. If you are running a previous version of WebSPIRS, back up anything you need to save and remove the old files using `pkgrm`.
4. Run `pkgadd` on the `webspirs.pkg` file.  

```
pkgadd -d ./webspirs.pkg
```

`pkgadd` modifies the `/etc/rc3.d/rc.local` file to start WebSPIRS at bootup and unpacks the WebSPIRS files into the following directory structure:

```
/opt/webspirs/bin  
/opt/webspirs/url/doc  
/opt/webspirs/url/images  
/opt/webspirs/template  
/opt/webspirs/template/help
```

`pkgadd` also places the `webspirs` link in the `/opt/httpd/cgi-bin` directory.



The installation script will ask a series of questions:

- What is the base directory?  
The default is `/opt/webspirs/`
- What is the base directory for your web server documents?  
`/usr/local/etc/httpd/htdocs/`
- What is the base directory for your web server and cgi scripts?  
`usr/local/etc/httpd/cgi-bin/`
- Enter a valid User ID. Example: `nobody`.  
Type `?` for a list of values.
- Enter a valid Group ID. Example: `nobody`.  
Type `?` for a list of values.

5. As described in the “Configuring WebSPIRS” section, edit the connections information in the `webspirs.cfg` file and the ERL server address information in the `erlclnt.cfg` file.

6. Start the WebSPIRS client from an XTerm to record the debug information

```
su nobody -c "/opt/webspirs/bin/webspird.cgi restart"
```

Or reboot your machine to start up the WebSPIRS processes automatically.

At this point, an end user can point a Web browser at the WebSPIRS client and perform searches. See the section, “Opening WebSPIRS On Your Browser.”

## Windows NT

Before you install WebSPIRS on a Windows NT platform, you should have the following minimum hardware requirements:

- 486/66 MHz IBM-compatible PC
- 32 MB RAM (1 to 10 simultaneous users). Add 2 MB per user over 10 users.
- Windows NT Workstation or Server Version 3.51+
- TCP/IP network protocol
- ERL server (separate machine)

To install WebSPIRS on Windows NT, complete the following steps:

1. Log in as `Administrator` on your machine.
2. Change to the directory to which you downloaded the files.
3. Execute the `webspirs.exe` file and then respond to the choices presented to you by the InstallShield Wizard, clicking `Next` to move through the script.
4. Read the Welcome screen message.
5. On the Setup Type screen choose a radio button for the Typical, Compact, or Custom setup type (Typical is the default). On this screen the directory where the files will be installed is shown. You can click `Browse` to select another choice.
6. On the Select Program Folder screen you can change the name from WebSPIRS to something else if you wish.
7. On the HTDOCS Directory screen click `Browse` to determine and then insert the location of your HTDOCS directory.
8. On the CGI-BIN Directory screen click `Browse` to determine and then insert the location of your CGI-BIN directory. Note that if you are using the WebSite server, this directory is called CGI-DOS.
9. On the CGI Executable Type page, select the type of executable files your HTTP server requires. Check the server documentation if you are not sure. Choose either the batch (`.BAT`) or command (`.CMD`) type.
10. On the Start Copying Files screen, your current settings are shown. If you are not satisfied with the settings, you can click the `Back` button to go back and make changes. After you click `Next` and the files copy, you will see a WebSPIRS Common Group of icons appear and another screen where you can access the release notes by clicking `Finish`.
11. Click the WebSPIRS icon in the WebSPIRS Common Group. A WebSPIRS MS-DOS window will appear displaying the message, "WebSPIRS started." Minimize this window.
12. Go to your browser and type in the URL address. See the section, "Opening WebSPIRS On Your Browser."

## Configuring WebSPIRS

Several files must be edited in order to configure WebSPIRS.

### The `webspirs.cfg` and `cgibaby.cfg` Files

The `/usr/local/etc/webspirs/bin/webspirs.cfg` file contains several configurable variables that affect the timing and connections in WebSPIRS.

#### Connections

Decide how many connections you want to make available through each copy of WebSPIRS and edit the following lines of the `webspirs.cfg` file (defaults are shown):

```
[WWW.DISPATCHER]
...
number_of_connections = 10 /* maximum number of cgiadult connections spawned.*/
[MM]
VM_REQUIRED=20000000
VM_DESIRED=20000000
```

Allow 2MB of memory for each connection.

#### Socket Port Number

Check the socket number in both the `webspirs.cfg` file and the `cgibaby.cfg` file. The `cgibaby.cfg` file's port number must match the `webspirs.cfg` file's port number. The `cgibaby` process finds the `cgichild` process by looking at the socket number. Note in the following line that the socket number is "6789."

```
[WWW]
request_name      :6789
```

If you are running two versions of WebSPIRS on the same machine, you must change the socket number.

#### Debug Value

To capture debug information when troubles occur, you must edit the `webspirs.cfg` file and change the line `DEBUG=0` to `DEBUG=1`.

#### Mail Activation

WebSPIRS provides the capability to mail search results records. However, this is a site-dependent function, and it is the responsibility of each site's WebSPIRS administrator to create a mail script and set the script in the `webspirs.cfg` file. Your platform must have a mail utility that can run on a command line and mail a file. You must name the script `spsmail` (or `spsmail.bat` for Windows). Here is a sample script for the Linux platform. This script deletes the file specified after it mails it:

```
mail $2 <$1
rm $1
```

You must change the script to an executable if it is a shell script. Set the executable permission as follows:

```
chmod u+x spsmail
```

When the command is run by WebSPIRS, it is invoked with two parameters. The first is the filename of the text to mail, and the second is the Email address filled in by the user.

If you are using the NT platform, you can download a copy of the Blat utility (<http://gepasi.dbs.aber.ac.uk/softw/blat.html>) for NT electronic mail. Put Blat and its .dll files in the bin directory along with the WebSPIRS executables. Check the Blat documentation and be sure to set it up properly so it can use your mail server. Here is a sample batch file (.bat) for the NT platform:

```
blat %1 -t %2 -s "WebSpirs Results"
del %1
```

After creating either of the above scripts, put it in the bin directory. Then set the script in the webspirs.cfg file:

```
sp.mailcmd="1"
```

After the sp.mailcmd variable is set to "1" in the webspirs.cfg file, you will see a "Mail" button on the Search page. Clicking this button will access the Mail Records page.

## The erlclnt.cfg File

The configuration file, /usr/local/etc/webspirs/bin/erlclnt.cfg, helps the WebSPIRS client establish the network connection to the ERL server. It contains the port information, protocol information, and the server identifier and address to which the workstation should connect. You can point to multiple servers by adding "server\_addrn=" lines, where "n" is an incremental number.

### Sample erlclnt.cfg file:

```
/* Creation date   :  Fr Nov 18 15:45:54 EST 1995
/* IP Address     :  192.80.71.114
server_addr1     =  /2/192.80.71.114/416
server_addrn     =  ...
```

The server address entry (server\_addr1) has meaning as described below:

- '2' indicates a TCP/IP protocol connection.
- '192.80.71.114' is the ERL server's IP address (a machine name such as 'skutter3.london.silverplatter.com' can also be used here).
- '416' is the port number assigned to all ERL communications traffic.

Insert the IP address or machine name of your ERL server in place of the default value.

Edit the ERL server address in the erlclnt.cfg file as shown below:

1. Open the erlclnt.cfg file using a text editor.
2. Modify the following TCP/IP address to match the address of your server.

```
server_addr1 = /2/192.80.71.114/416
```

If you have more than one server, create a line for each server, incrementing the n with each one:

```
server_addr1 = /2/192.80.71.114/416
server_addrn =
```

3. Save the erlclnt.cfg file in text-only format.

## The .htaccess File

It is important to restrict access to WebSPIRS by creating a `.htaccess` restriction file. To create this file, complete the following steps:

1. Mimic the following example file:

```
AuthUserFile /dev/null
AuthGroupFile /dev/null
AuthName InternalOnly
AuthType Basic
<Limit GET>
order deny,allow
deny from all
allow from silverplatter.com
#allow from 192.80.71
</Limit>
```

2. Add an "allow" line with the name or the IP address for only those domains you wish to have access. For example, you can replace the `silverplatter.com` domain with your own domain in the line "allow from `silverplatter.com`." Only allowed domains can access the WebSPIRS files. Note that the `silverplatter.com` domain has been allowed and the `192.80.71` IP address has been commented out in the above sample `.htaccess` file.
3. Put the `.htaccess` restriction file in the `cgi-bin` directory.

## The mime.types File

You should also edit the `mime.types` file so that WebSPIRS can recognize files with both the `.html` and `.htm` file extensions. You will find this file in the `/usr/local/etc/httpd/conf` directory. Edit the following line in that file:

```
text/html      html
```

So that it reads:

```
text/html      html htm
```

## Opening WebSPIRS On Your Browser

For the WebSPIRS ERL gateway client to be usable, the WebSPIRS process must be running and an end user must have a Web browser pointed at the WebSPIRS gateway.

1. Enter the following destination in your Web Browser:

```
1. http://your_machine_name/webspirs/webspirs.htm
```

where `your_machine_name` is the IP address or machine name of the Web server running WebSPIRS. Note that you may use "localhost" if your browser and server are running on the same machine.

2. Enter your username and password.

Use the available on-line help for assistance in formulating search statements and displaying results.



---

## Chapter 3 - Tutorial for Customizing Templates

---

This tutorial for customizing WebSPIRS templates answers the following questions:

- What are the WebSPIRS templates?
- What do the encoded macros do?
- Why customize a template?
- How can I edit a template?
- What kinds of customizations can I make?

This chapter also includes a diagram of the main WebSPIRS templates and a description of each template and fragment.

### What are the WebSPIRS templates?

The WebSPIRS templates generate the interface pages which the user sees. They are composed of HTML formatted text encoded with a macro language understood by WebSPIRS. The templates can be found in the `/webspirs/template` directory. When a page is requested, WebSPIRS parses the requested template (`.htm` file) and the macros expand. The page is then fed back to the user. The templates are processed into forms by the `cgiaadult` process, which is described in Chapter 1, "Overview."

### What do the encoded macros do?

The macros encoded in the templates contain the contents of the web forms, log the user in to ERL, and perform such tasks as opening and searching databases. These predefined macros are preceded by the `[SP_MACRO]` tag and are followed by the `[/SP_MACRO]` tag.

Following is an example of a macro used in the `database.htm` template, which inserts a list of databases with checkboxes preceding the title of each database:

```
[SP_MACRO] sp.avail.dbs.foreach normalfile="dblitmck.htm"
titlefile="dblttlck.htm"[/SP_MACRO]
```

Before customizing or creating new templates, you should understand both the existing templates and the macros encoded in them. A diagram showing the flow of the WebSPIRS templates and a description of each template is provided in the section, "WebSPIRS Templates." For more details on the SilverPlatter macros and a description of each macro, see the section, "Understanding the Macros" in Chapter 4.

### Why Customize a Template?

There are many possible reasons for customizing a template. You could customize a template to make it more specific to an often-used database. For example, if you always use the ERIC database and often search identical fields, you could make modifications to eliminate unnecessary steps. You could bypass the Login and Database pages and create a new Search form entitled, "Search ERIC," which displays after the user logs in. You can edit the `search.htm` template (the template that allows the user to create Boolean searches) to create the Search ERIC form. You can further customize the template so that the form is limited to searching specific fields, such as author (AU) and title (TI).

You can also personalize the interface by adding your company's name, logo, and other identifying information. You can even add hyperlinks to provide additional in-depth marketing information if you wish. By using the `head.htm` and `foot.htm` templates you can create a consistent look for each page. You can add information such as graphics and logos at the beginning or end of each page, and you can select a consistent background color or graphic for the page.

## How can I edit a template?

You can edit an existing template to create a new template. The templates are created in HTML, and you should have some familiarity with HTML before you begin. The Library of Congress maintains a web page (<http://lcweb.loc.gov/global/internet/dev.html>) of Internet resources which can help you get started. You will need a text editor to make the changes to the template files. You can edit templates using any text editor on any platform. Two WebSPIRS directories contain the files you will need:

- bin
- templates

In the bin directory, you will find a file called `webspircfg`. Edit this file so that the path of "template\_directory" is the same as the directory location of the templates on your system. The default is `../template`, which should work on most installations.

After you have created a new template, be sure it is in the same directory as the other templates. The `head.htm` template has variables for standard template names. You may change these variables to other names. See the explanation of macro variables in the section "Understanding the Macros" in Chapter 4.

## The Editable and Processed Template States

Before a template is processed into the source for a Web browser, it can be edited. Following is an example of the editable `login.htm` template, which will be processed into the WebSPIRS login page. Note the SilverPlatter macro tags:

```
<!-- Main page: Login prompt -->
[SP_MACRO]sp.assign reset="yes" macro="sp.template_description" value="Login to
WebSPIRS" [/SP_MACRO]
[SP_MACRO]sp.include filename=sp.form.head.p [/SP_MACRO]
<H1><IMG SRC="[SP_MACRO]sp.webspircs.imagedir[/SP_MACRO]webabout.gif"
ALT="Welcome to " align=center> <IMG SRC="[SP_MACRO]sp.webspircs.imagedir[/SP_MACRO]
webspircs.gif" ALT="WebSPIRS" align=center></H1>
[SP_MACRO]sp.include filename="news.htm" [/SP_MACRO]
<!-- If error message is set, show it here -->
[sp_macro]sp.include interpret="yes"
type="STRING" leftvalue="" condition="NE" rightvalue="sp.login_error"
inclusion=[sp_block]
  <hr size=3>
  <strong>
<h2>Login:</h2>
  <INPUT TYPE="hidden" NAME="sp.nextform" VALUE="[SP_MACRO]sp.nextform[/SP_MACRO]">
  <P>User name: <INPUT TYPE="text" NAME="sp.username" SIZE="31" MAXLENGTH="31"
  VALUE="[SP_MACRO]sp.username.suggest[/SP_MACRO]">
  <br>Password: <INPUT TYPE="password" NAME="sp.password" SIZE="31" MAXLENGTH="31"
  VALUE="[SP_MACRO]sp.password.suggest[/SP_MACRO]">
  <INPUT TYPE="submit" NAME="submit" VALUE="Login"> <INPUT TYPE="reset" NAME=""
  VALUE="Reset"></p>
</FORM>
[SP_MACRO]sp.include filename=sp.form.foot.p [/SP_MACRO]
```



The WebSPIRS code reads the editable template and it becomes the source for the Web browser. For example, when WebSPIRS processes the following macro:

```
[SP_MACRO]sp.assign reset="yes" macro="sp.template_description" value="Login to
WebSPIRS" [/SP_MACRO]
```

the macro sets the value of the variable `sp.template_description`. The `head.htm` template uses that value to fill in the page title, and the `foot.htm` template uses the value to fill in the page name. Then in the processed template it becomes

```
<TITLE>WebSPIRS: Login to WebSPIRS</TITLE>
```

Following is the complete processed template:

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0 Strict Level 2//EN">
<HTML>
<HEAD>
<TITLE>WebSPIRS: Login to WebSPIRS</TITLE>
</HEAD>
<BODY BACKGROUND="/webspirs/doc/paper.jpg">
<H1>Welcome to <IMG SRC="/webspirs/doc/webspirs.gif" ALT="WebSPIRS"></H1>
<P><STRONG>Login:</STRONG></P>
<FORM METHOD="POST" ACTION="http://nova.silverplatter.com/cgi-bin/webspirs.cgi">
  <INPUT TYPE="HIDDEN" NAME="DEFAULT.SP.FORM.HEAD.P" VALUE="head.htm">
  <INPUT TYPE="HIDDEN" NAME="DEFAULT.SP.FORM.FOOT.P" VALUE="foot.htm">
  <INPUT TYPE="HIDDEN" NAME="SP.USERNUMBER.P" VALUE="8">
  <INPUT TYPE="HIDDEN" NAME="SP.FORM.SEARCH.P" VALUE="search.htm">
  <INPUT TYPE="HIDDEN" NAME="SP.FORM.TOP.P" VALUE="top.htm">
  <INPUT TYPE="HIDDEN" NAME="SP.THISFORM" VALUE="login.htm">
  <INPUT TYPE="hidden" NAME="sp.nextform" VALUE="top.htm">
<P>User name: <INPUT TYPE="text" NAME="sp.username" SIZE="16" MAXLENGTH="16"
  VALUE="guest">
<br>Password: <INPUT TYPE="password" NAME="sp.password" SIZE="16" MAXLENGTH="16"
  VALUE="guest">
<P><INPUT TYPE="submit" NAME="submit" VALUE="Login"> <INPUT TYPE="reset" NAME=""
  VALUE="Reset">
</FORM>
<HR>
<ADDRESS>
<A HREF="http://www.silverplatter.com/">
<IMG ALIGN=bottom SRC="/webspirs/doc/sp35.gif" ALT="SilverPlatter World"></A>
Copyright 1996, SilverPlatter International NV,
<A HREF="/webspirs/doc/start.htm">WebSPIRS</A>Version 3.0
<BR>Send your comments to
<A HREF="mailto:webspirs@silverplatter.com">WebSPIRS@SilverPlatter.com</A>,
referencing "Login to WebSPIRS".
</ADDRESS>
</BODY>
</HTML>
```

## The FORM in WebSPIRS

In the two states of the WebSPIRS login template shown above, you will see <FORM> tags. In WebSPIRS the `action.htm` file is included in the template by the `sp.include` macro. It generates the <FORM ACTION=...> data to link the template back to WebSPIRS.

```
<FORM [SP_MACRO]sp.include filename="action.htm"[/SP_MACRO]>
```

When the form's submit button is clicked by the user, the form is processed and a request is submitted to the web server. A form has an ACTION and METHOD parameter. The ACTION parameter specifies which uniform resource locator (URL) the form data should be sent to for further processing. In the above example from the editable template, the METHOD="POST" statement is not visible; it is contained in the `action.htm` file:

```
METHOD="POST" ACTION="http://[SP_MACRO]server_name[/SP_MACRO]
[SP_MACRO]script_name[/SP_MACRO]"
```

It becomes visible in the processed template state. The POST method formats the data and sends the form data as one long text string. It calls for a URL, which is an executable program in the `cgi-bin` directory. When the user clicks on the submit button, the FORM details are written out into a file (`wwwlast.req`) and the ACTION URL is called.

Note: Changing `action.htm` is not recommended.

## What kinds of customizations can I make?

This section will help you perform the following tasks. Note that you are not limited to these customizations:

- Bypassing the Login page
- Preselecting Databases
- Displaying specific fields
- Changing the records display default number
- Changing a template title
- Changing the graphical interface
- Creating a table
- Controlling the flow of pages

### Bypassing the Login page

The best way to bypass the Login page is to set the username and password for each user in a separate `head.htm` file; for example, in a file called `billhead.htm` you would add the following lines:

```
[SP_MACRO]sp.assign macro="sp.username" value="bill"[/SP_MACRO]
[SP_MACRO]sp.assign macro="sp.password" value="<password>"[/SP_MACRO]
```

There are other ways of bypassing the Login page. You can set the username and password in the `webspirs.cfg` file, for example:

```
sp.username=name
sp.password=password
```

You can also pass the username and password in the command line and then save it as a bookmark if you are using Netscape:

```
http://myserver/cgi-bin/webspirs.cgi?sp.username=name&sp.password=pw
```

The disadvantage to this is that the password can be seen.

## Preselecting Databases

You can preselect databases and present the user with a search page with databases already selected for them. This is useful if the users always search the same databases. You can set the database ID or the set ID using the methods described in "Bypassing the Login Page." For example, you can set an individual Medline database or set of Medline databases in the `head.htm` file by adding one of the following lines:

```
[SP_MACRO]sp.assign macro="sp.dbid" value="ML4L" [/SP_MACRO]
[SP_MACRO]sp.assign macro="sp.setid" value="ML" [/SP_MACRO]
```

You can add one of the following lines to the `webspirs.cfg` file as follows:

```
sp.dbid=ML4L
sp.setid=ML
```

You can pass the database ID in the command line and save it as a bookmark in Netscape:

```
http://myserver/cgi-bin/webspirs.cgi?sp.dbid=ML4L
```

## Displaying Specific Fields

If you want to consistently display specific fields in your search results, such as the title (TI) and author (AU) fields, you can make a simple change in the `head.htm` file to change the default fields to display. Edit the following line in that template:

```
[sp_macro]sp.assign macro="sp.record.fields.p" value="*URLFIELD" [/sp_macro]
```

Change the `value=` parameter to `"TI,AU"`. The line will then read:

```
[sp_macro]sp.assign macro="sp.record.fields.p" value="TI,AU" [/sp_macro]
```

You can set WebSPIRS to display other specific fields as well. To display brief fields (citation fields) as the default, use `"CITN"` for the `value=` parameter. The `CITN` fields vary according to database. For example, for the PsycLIT Journal Articles database the fields are Title, Author, Institutional Affiliation of First Author, Journal Name, and the abstract volume and number. Of course, by selecting "Brief Fields" from the Search page, you will get the same result. This is the simplest way, but it will only last for the current session.

Note: If you are interested in displaying fields other than "Brief Fields" for only the current session, you can access the Record Display Options page by clicking the "Options" button on the Search page. Then you can select which fields to display and how they are to be sorted.

## Changing the Records Display Default Number

If you would like more than five records to display on your page of search results, you can change the default number of records by making a simple edit in the `head.htm` template. Edit the following line in that file:

```
[sp_macro]sp.assign macro="sp.record.howmany.p" value="5"[/sp_macro]
```

Change the `value=` parameter to another number; for example, you could change it to "20" so that the line reads:

```
[sp_macro]sp.assign macro="sp.record.howmany.p" value="20"[/sp_macro]
```

You can use any number no matter how high for the `value=` parameter.

## Changing a Template Title

At the top of each complete WebSPIRS template is a line similar to the following line from the `search.htm` template:

```
[SP_MACRO]sp.assign macro="sp.template_description" value="Search"[/SP_MACRO]
```

In this line the `sp.assign macro` is used to assign a value to the macro variable `sp.template_description`. The value is the template name, in this case "Search." When the resulting page from this template is displayed by your browser, the word "Search" will appear at the top of the browser page. For example, if you use Netscape, the words **Netscape - [WebSPIRS: Search]** will appear in the Netscape title bar. (The template title also appears in the footer as a reference page name.)

By changing the `value=` parameter for the `sp.template_description` variable, you change the title of the template.

## Changing the Graphical Interface

This section provides instructions for customizing the following graphical aspects of the WebSPIRS pages:

- The Toolbar
- Background Graphic
- Logo Graphic

**Note:** The Library of Congress provides access to helpful "Icons, Images, and Backgrounds for WebPages" at the following URL:

```
http://lcweb.loc.gov/global/internet/html.html#graphics
```

## The Toolbar

At the top of each WebSPIRS page (except the Login and Logout pages) you will find the toolbar icons. For example, the Search page displays the toolbar icons. That page is generated from the `search.htm` template, which allows the user to create Boolean searches. The following lines from that template set the appropriate appearance and connections for the toolbar. Since this is the Search page, the Search icon is set to the on state and the Help icon points to the appropriate help page.

```
[SP_MACRO] sp.assign macro="sp.tbar.image.search" value="srchon.gif"
           reset="y" [/SP_MACRO]
[SP_MACRO] sp.assign macro="sp.tbar.page.help" value="c_srch.htm"
           reset="y" [/SP_MACRO]
[SP_MACRO] sp.include filename="tbar.htm" [/SP_MACRO]
[SP_MACRO] sp.assign macro="sp.tbar.image.search" value="srchoff.gif"
           reset="y" [/SP_MACRO]
[SP_MACRO] sp.assign macro="sp.tbar.page.help" value="help.htm"
           reset="y" [/SP_MACRO]
```

The `tbar.htm` template file is included to add the toolbar to the page. The `sp.assign` macro is used four times, twice to assign values to the `sp.tbar.image.search` variable and twice to assign values to the `sp.tbar.page.help` variable. The graphic file, `srchon.gif`, provides the Search toolbar icon in the on state (with a "pushed in" appearance). The graphic file, `srchoff.gif`, provides the Search toolbar icon in the off state. The `c_srch.htm` file provides on-line help for searching, and the `help.htm` file provides the following basic help message, "A help topic was not specified. Please contact your administrator and report this problem," if a help topic is not found.

By editing the `head.htm` file, you can substitute your own graphic images for the toolbar icons. This is done by changing the filenames. Or you can edit the files and keep the names. No template changes are necessary. You can also change the name of the help file for this page and move the toolbar to a different location on the page.

**Note:** In the `webspirs.cfg` file the `head.htm` file is set to the `sp.form.head.p` macro variable.

## Background Graphic

You can change the background for each WebSPIRS page by editing the `head.htm` template and changing the graphic file. For example, in the following line taken from `head.htm`, you can substitute a different graphic image for the `paper.jpg` file which provides the gray, paper look of each WebSPIRS page:

```
<BODY BACKGROUND="[SP_MACRO]sp.webspirs.imagedir[/SP_MACRO]paper.jpg">
```

## Logo Graphic

The SilverPlatter logo is shown at the bottom of each WebSPIRS page. The graphic file for the logo is included in the `foot.htm` template, which provides a consistent look for the bottom of each page. You can change the graphic file by substituting a different graphic image file for the `sp.35.gif` file in the following line from `foot.htm`:

```
<IMG border=no ALIGN=left SRC="[SP_MACRO]sp.webspirs.imagedir[/SP_MACRO]sp35.gif"
      ALT="SilverPlatter World">
```

If you substitute your own file, you will probably want to provide a different value for the `ALT=` parameter as well.

## Creating a Table

You can organize some of the information on the pages into tables. For example, on the UBP WebSPIRS client, the account balance information appears as one sentence on the Search page. Here is the HTML format in the `tbar.htm` template:

```
<p><b>Charges for [sp_macro]sp.username[/sp_macro];</b>
This session <b>${[sp_macro]sp.admin.total.used.[/sp_macro]}</b>
Account balance <b>${[sp_macro]sp.admin.balance[/sp_macro]}</b>
```

After WebSPIRS expands the macros, it displays as one sentence on the Search page as follows:

**Charges for bettyk: This session \$0.00; Account balance \$263.60.**

By adding table tags to the `tbar.htm` template, you can encompass this information in a 3-column table:

```
<p><TABLE BORDER>
<TR VALIGN= "MIDDLE">
<TD>
<B>Charges for [sp_macro]sp.username[/sp_macro]:</B></TD>
<TD>
This session <B>${sp_macro}sp.admin.total.used[/sp_macro]</B></TD>
<TD>
Account balance <B>${sp_macro}sp.admin.balance[/sp_macro]</B></TD>
</TR>
</TABLE>
```

After adding the table tags to the `tbar.htm` template, this information will appear on the Search page in a three-column table similar to the following:

<b>Charges for bettyk:</b>	This session <b>\$0.00</b>	Account balance <b>\$263.60</b>
----------------------------	----------------------------	---------------------------------

## Controlling the Flow of Pages

The user advances through the pages of WebSPIRS by way of two macros that display the next form. One is the `sp.nextform` macro variable, and the other way is the `sp.generate_url` macro command. There are two basic ways to implement these macros:

- Using a Submit Button
- Using a URL

### Using a Submit Button

The most common way of controlling the flow of pages is through the use of the `sp.nextform` variable. You can set the variable to the next page by using a submit button. Here is an example (note that the space after `index.htm` is required):

```
<INPUT NAME="sp.nextform=index.htm " TYPE="SUBMIT" VALUE="Index">
```

The above example sets the `sp.nextform` variable to the `index.htm` template causing that page to load next. You can add a list of `tag=values` to the `sp.nextform` parameter as well. The `VALUE=` parameter is the label for the button. You can name a submit button anything you wish by changing that parameter. If you omit the `VALUE=` parameter, the button will be labeled "Submit."

### Using a URL

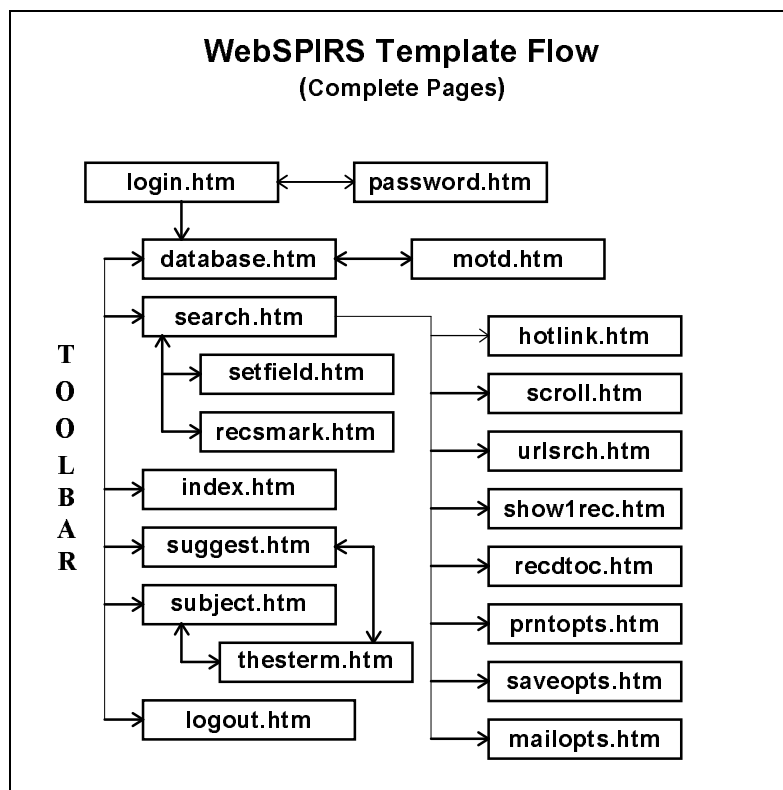
Another method of moving to the next form is by using the `sp.generate_url` macro command. Here is an example:

```
[SP_MACRO]sp.generate_url nextform="thesterm.htm"
args="sp.thesaurus.term.p=sp.currterm" [/SP_MACRO]
```

The macro generates a URL to another template, in this case to the Thesaurus page template. The `nextform=` parameter specifies the template to be loaded if the URL is followed. The `args=` parameter specifies arguments to be added onto the URL, in this case it specifies the current term.

## WebSPIRS Templates

This section describes the existing WebSPIRS templates. The following diagram shows the interaction of those WebSPIRS templates which provide the complete pages. There are many additional large fragments and utility fragments.



### Complete Pages

Following is an alphabetical list of the WebSPIRS templates which are complete pages. These pages may use fragments, but they are complete unto themselves:

<b>database.htm</b>	This template allows users to select a database from a list by clicking on a checkbox.
<b>hotlink.htm</b>	This template is used if the record is hotlinked. It displays the linked to record. In this template, the <code>sp.record.source.p</code> macro is assigned a value of " <code>_HOTLINK</code> " to indicate that the record is to come from a hotlink.
<b>index.htm</b>	This template is accessed from the Index button on the toolbar. The user can choose to view the records for a selected index term(s) and can change the viewed index.
<b>login.htm</b>	This template provides the WebSPIRS login page.
<b>logout.htm</b>	This template allows the user to log out of WebSPIRS and free the connection.

---

<b>mailopts.htm</b>	This template provides the Mail Records page which is accessed when the "Mail" button is clicked on the Search page.
<b>motd.htm</b>	This template provides the Message of the Day from the WebSPIRS system administrator. It is accessed by way of a hotlink on the <code>database.htm</code> page.
<b>password.htm</b>	This template provides the "Change Password" page which allows the user to change his password.
<b>prntopts.htm</b>	This template provides the Print Records page which is accessed when the "Print" button is clicked on the Search page. It connects to the <code>exphtml.htm</code> template.
<b>recdtoc.htm</b>	This template provides a table of contents for some full-text records (by way of a hyperlink).
<b>recprint.htm</b>	This template provides the current records in a format which will look better when printed; that is, it will not contain form controls or the toolbar.
<b>recsmark.htm</b>	This template shows the records which the user has marked for later printing or saving to a file. The user can also unmark records.
<b>saveopts.htm</b>	This template provides the Save Records page, which is accessed when the "Save" button is clicked on the Search page. It connects to the <code>exprow.htm</code> template.
<b>scroll.htm</b>	This template provides a more convenient display of records after the user has scrolled them.
<b>search.htm</b>	This template allows the user to build a Boolean search and create a search history by including the <code>srchconn.htm</code> fragment. The records are displayed on the page by including the <code>rechits.htm</code> template fragment.
<b>setfield.htm</b>	This template provides provides the page shown when the user clicks the Options button on the Search page ( <code>search.htm</code> ). The user can select specific fields from the list.
<b>show1rec.htm</b>	This template displays all the fields in a single record from a hotlinked record title.
<b>subject.htm</b>	This template is accessed from the Thesaurus button on the toolbar. It provides the Thesaurus Subject List.
<b>suggest.htm</b>	This template is accessed from the Suggest button on the toolbar. It allows the user to enter a word or phrase to be matched with a database's controlled vocabulary. A list of suggested terms displays.
<b>thesterm.htm</b>	This template provides the term details for a thesaurus term selected from the Thesaurus Subject List or the Suggest List.
<b>urlsrch.htm</b>	This template composes searches from hotlinks accessed from the index or history.



## Large Fragments

Following are large fragments used by other templates:

<b>rechits.htm</b>	This fragment displays the retrieved records. It is included in <code>search.htm</code> and <code>scroll.htm</code> .
<b>seldbs.htm</b>	This fragment provides the list of selected databases. The user can deselect a database by clicking the checkbox before the database name. It is used on most templates.
<b>srchcomn.htm</b>	This fragment builds the limit inputs and field lists and retrieves the records for a search. It is included in the <code>search.htm</code> and <code>scroll.htm</code> templates.

## Utility Fragments

Following are utility fragments used by the templates:

<b>action.htm</b>	This fragment file is included to generate the <code>&lt;FORM ACTION=...&gt;</code> data to link back templates to WebSPIRS. It is used on every form page after <code>&lt;FORM</code> .
<b>checkbox.htm</b>	This fragment adds a checkbox. It is used by subheadings, the database list, and for individual purposes.
<b>clsrch.htm</b>	This fragment clears the search fields and displays the search page again.
<b>dbitem.htm</b>	This fragment provides the format for each item in the opened database list. It is used in <code>seldbs.htm</code> .
<b>dblitmck.htm</b>	This fragment provides each item in the available database list. It is used in <code>database.htm</code> .
<b>exphtml.htm</b>	This fragment shows records and the search history in HTML format. It does not show the buttons.
<b>expmail.htm</b>	This fragment includes either <code>exphtml.htm</code> or <code>expraw.htm</code> depending upon the <code>sp.export.format</code> set by the user in <code>mailopts.htm</code> .
<b>expraw.htm</b>	This fragment shows records and the search history minus the HTML markup and buttons.
<b>expterm.htm</b>	This fragment explodes a thesaurus term for searching and returns to the search page.
<b>fielditm.htm</b>	This fragment provides the format for each field in a field list, with a checkbox. Used in <code>setfield.htm</code> .
<b>foot.htm</b>	This fragment provides a consistent look for the bottom of each WebSPIRS page and is included at the bottom of each template.
<b>head.htm</b>	This fragment provides a consistent look for the top of each WebSPIRS page and is included at the top of each template. Many variables are set once in this template.
<b>indterm.htm</b>	This fragment provides the format for an index list term.
<b>maildone.htm</b>	This fragment provides the page that displays after mail has been sent. It shows the message, "Mail sent! Your records have been mailed."

<b>mrkclear.htm</b>	This fragment clears marked records and displays all the records again.
<b>news.htm</b>	This fragment provides a news area for the login page. Edit this template if you want to provide a message for others to see before they log in.
<b>pagesize.htm</b>	This fragment provides a drop down list of the number of records to show. You can edit this template to limit or increase the number of records users can see at one time.
<b>recfmt.htm</b>	This fragment provides the format for each record with no checkbox.
<b>recfmtck.htm</b>	This fragment provides the format for each record with a checkbox for marking.
<b>recfmtrw.htm</b>	This fragment provides the record output in raw (no HTML; no buttons) format.
<b>srchterm.htm</b>	This fragment searches for a thesaurus term and returns to the search page.
<b>subtermd.htm</b>	This fragment provides the format for a term description from a suggested list of terms.
<b>sugsrch.htm</b>	This fragment provides a search based on the term from a suggested list of terms.
<b>sugterm.htm</b>	This fragment provides the format for a suggested list term.
<b>tbar.htm</b>	This fragment provides the WebSPIRS toolbar, which is found on all main pages except the login and logout pages. The names of the <code>.gif</code> files are set in <code>head.htm</code> .
<b>thesitem.htm</b>	This fragment provides the format to make the thesaurus term into a link to its term detail.
<b>top.htm</b>	This fragment provides the first "real" page in the system. The user does not see this page. It allows a one-time-per-session setting of the usage based pricing (UBP) information.

---

## Chapter 4 - Creating and Using Macros

---

This chapter defines all of the SilverPlatter macro commands and macro variables. These macros are listed under the following categories:

- Interface-specific variables
- Administration
- General database
- Text display
- Field-specific index
- Search
- Automatic subject lookup
- Field list
- Thesaurus term
- Guide keyword
- Marked records
- Miscellaneous macro support

You can add macros of your own design by following the instructions in the “Adding Macros” section.

### Understanding the Macros

Many predefined macros are contained in the HTML templates used by WebSPIRS. These macros are preceded by the `[SP_MACRO]` tag and followed by the `[/SP_MACRO]` tag. By using

```
[SP_MACRO] sp.assign macro="sp.template.description" value="..."[/SP_MACRO]
```

the contents of the `value=` parameter can be placed in the macro variable set by the `macro=` parameter. See the section “Miscellaneous Macro Support” for details about the `sp.assign` macro.

Macros can be used as names for any form widget. New macros can be added to the server code without changing the code by implementing a new class derived from **www\_HTML\_Helper** and then relinking the server. The macros depend upon the Content Operative Retrieval Engine (CORE ) source code.

The SilverPlatter macros help the user log in to ERL, move through the web forms, open and search databases, retrieve and display text, scroll through a selected list of terms, access database-specific help, mark records, and lookup subjects automatically.

There are two types of macros:

- **Macro commands** - These macros cause an action to be performed and are generally coded in the template. Many macro commands will cause HTML output to be included. Examples of macro commands are `sp.assign`, which generates no output, and `sp.record.text`, which generates text for records.
- **Macro variables** - These macros have a value and are generally swapped back and forth from the client to the server as HTTP "macro=value" pairs. Examples of macro variables are `sp.search.value.p`, which is used as the search string, and `sp.dbid.p`, which indicates the opened database. Some variables are defined by WebSPIRS to indicate options, such as `sp.search.value.p`. Other variables are used by templates and are not used internally by WebSPIRS, such as `sp.back.form.p`. You may freely create any variables for your own use by way of the `sp.assign` macro. See “Miscellaneous Macro Support” for details on the `sp.assign` macro.

Some macro variables have a **.p** or a **.pp** suffix. The suffix **.p** indicates persistence, which means it exists throughout the active session, and it writes out the variable as long as there is no `<INPUT>` or `<SELECTED>` statement for the name. When using multiple values that may or may not be displayed together, the **.pp** suffix writes out the variable if it doesn't find the value in one of the `<INPUT>` statements.

## Nested Macros

Macros can be nested and can be included as arguments to other macros using the `[sp_block]...[/sp_block]` tags. In the following example, which was taken from the `search.htm` file, the `sp.assign` macro has been nested within the `sp.foreach` macro and is the `text=` argument:

```
[SP_MACRO]sp.foreach iterator="LIST" list=sp.search.field2.p variable="sp.search.field"
  text=[sp_block][sp_macro]sp.assign macro="sp.search.2" append="yes" interpret="yes"
  value="({{sp.search.term2.p}} { in } sp.search.field (sp.ifnotlast))"[/sp_macro]
[/sp_block]
  ifnotlast=" or "
  default=[sp_block][sp_macro]sp.assign macro="sp.search.2"
  value=sp.search.term2[/sp_macro][[/sp_block]
[/SP_MACRO]
```

## SilverPlatter Macros

This section describes all of the SilverPlatter macro commands and macro variables available for use.

### Interface-Specific Variables

The following interface macro variables help the user log in to WebSPIRS and navigate through the pages:

---

#### **sp.username**

Example: `[SP_MACRO]sp.username[/SP_MACRO]`

The value of this macro variable is the name of the user accessing the ERL account. This is found on the `login.htm`, `tbar.htm`, and `password.htm` templates.

---

#### **sp.password**

Example: `[SP_MACRO]sp.password[/SP_MACRO]`

The value of this macro variable is the password of the user accessing the ERL account. This is found on the `login.htm` and `password.htm` templates.

---

#### **sp.thisform**

Example:

```
[SP_MACRO]sp.assign macro="sp.thisform"
  reset="yes" interpret="yes" value="sp.form.search.p"[/SP_MACRO]
```

The value of this macro variable is the template that created the current form. It is used on the `rechits.htm` template and many other templates.

### **sp.nextform**

Example: `[SP_MACRO] sp.nextform[/SP_MACRO]`

The value of this macro variable is the template to use for the next form. It is found on most templates.

---

### **sp.search.invalid\_message**

Example:

```
[SP_MACRO] sp.if left value = "" condition="NE"
right value="sp.search.invalid_message" [/SP_MACRO]
```

The value of this macro variable is the message generated if there is an error in a search, such as a parsing error. It is found on the `srchconn.htm` template.

---

### **sp.form.show.p**

Example:

```
[SP_MACRO] sp.assign macro="sp.form.show.p"
value="search.htm" [/SP_MACRO]
```

This macro variable holds the name of the default record display template. It is used on the `head.htm` and `rechits.htm` templates.

---

### **sp.form.search.p**

Example:

```
[SP_MACRO] sp.assign macro="sp.form.search.p"
value="search.htm" [/SP_MACRO]
```

This macro variable holds the name of the default search template.

---

### **sp.login\_error**

Example: `[SP_MACRO] sp.login_error[/SP_MACRO]`

This macro variable inserts an error message if the login information is incorrect. It is used in `login.htm`.

---

### **sp.template\_description**

Example: `[SP_MACRO] sp.template_description value="Database Search" [/SP_MACRO]`

The value of this macro variable is the name or title of the template, for example, "Database Search." It is found on all main templates.

---

### **sp.form.head.p**

Example: `[SP_MACRO] sp.include filename=sp.form.head.p[/SP_MACRO]`

This macro variable is the top of the form. It is found on all main templates.

---

**sp.form.foot.p**

Example: `[SP_MACRO]sp.include filename=sp.form.foot.p[/SP_MACRO]`

This macro variable is the bottom of the form. It is similar to `sp.form.head.p` in that it appears on all form templates.

---

**sp.form.top.p**

Example: `[SP_MACRO]sp.form.top.p[/SP_MACRO]`

This macro variable holds the name of the default database list template.

---

**sp.back.form.p**

Example:

```
[SP_MACRO]sp.assign macro="sp.back.form.p"
reset="yes" value=sp.thisform[/SP_MACRO]
```

The macro variable `sp.back.form.p` is assigned a value by the `sp.assign` macro. If the page is the Search page, it has a value of the `search.htm` file. It is found on most templates.

---

**sp.back.form.title.p**

Example:

```
[SP_MACRO]sp.assign macro="sp.back.form.title.p"
reset="yes" value="Back to search"[/SP_MACRO]
```

The macro variable `sp.back.form.title.p` is assigned a value by the `sp.assign` macro. The value is the title of the back button. In the case of the above example, the title (value) is "Back to search." It is found on most templates.

---

**sp.tbar.page.value**

Example:

```
[SP_MACRO]sp.assign macro="sp.tbar.page.search"
value="search.htm" reset="y"[/SP_MACRO]
```

This macro variable provides the page accessed from a toolbar icon, such as the Search, Database, Index, Suggest, Thesaurus, Help and Logout icons. It is found on the `head.htm` and `tbar.htm` templates.

---

**sp.export.range.p**

Example:

```
[SP_MACRO]sp.assign macro="sp.export.range.p"
value="PAGE" [/SP_MACRO]
```

This macro variable specifies what to export according to the value ("PAGE", "ALL", or "MARKED"). It is used in the `prntopts.htm`, `saveopts.htm`, and `mailopts.htm` templates. An example of its use can be seen on the Save Records page, where the user can save the records on the previous page, all the records, or the marked records.

**sp.export.save.history.p**

Example:

```
[SP_MACRO]sp.if condition= "EQ"  
leftvalue= "" rightvalue= sp.export.save.history.p  
then=[sp_block]...[/sp_block] [/SP_MACRO]
```

This macro variable includes the search history in the output when printing, saving, or mailing records. It is used on the `exphtml.htm`, `expdraw.htm`, `prntopts.htm`, `saveopts.htm`, and `mailopts.htm` templates.

**sp.export.recnums.p**

Example:

```
[SP_MACRO]sp.if leftvalue="1"  
condition="EQ" rightvalue=sp.export.recnums.p  
then=[sp_block]...[/sp_block] [/SP_MACRO]
```

This macro variable includes the number of records in the output when printing, saving, or mailing records. It is used in the `recfmt.htm`, `prntopts.htm`, `saveopts.htm`, and `mailopts.htm` templates.

---

**sp.export.format.p**

Example:

```
[SP_MACRO]sp.if leftvalue="raw"  
condition="EQ" rightvalue=sp.export.format.p  
then=[sp_block]...[/sp_block] [/SP_MACRO]
```

This macro variable determines whether the output will be "raw" (no HTML; no buttons) or "html" (HTML but no buttons). If "raw", `exprow.htm` is used and if "html", `exphtml.htm` is used. It is found on the `expmail.htm` and `mailopts.htm` templates.

---

**sp.export.mailto.p**

Example: `[SP_MACRO]sp.export.mailto.p[/SP_MACRO]`

This macro variable is passed to the `sp.mailcmd` mail script as the user's Email address. It is used on the `mailopts.htm` template.

## Administration Macros

The following macro commands and macro variables relate to WebSPIRS and ERL administration:

### Macro Commands

The following are administration macro commands:

---

#### sp.webspirs.version

Example: `[SP_MACRO]sp.webspirs.version[/SP_MACRO]`

This macro defines the current version of WebSPIRS. It is also used in `foot.htm`, and it adds the WebSPIRS version number. It is set in the `webspirs.cfg` file.

```
Version [SP_MACRO]sp.webspirs.version[/SP_MACRO].
```

---

#### sp.erl.logout

Example: `[SP_MACRO]sp.erl.logout[/SP_MACRO]`

This macro disconnects the user from the ERL server(s).

---

#### sp.erl.message.of.the.day

Example: `[SP_MACRO]sp.erl.message.of.the.day[/SP_MACRO]`

This macro provides the message of the day from the ERL administrator if a message exists. It is found in the `motd.htm` and the `database.htm` template. Here is an example:

```
[SP_MACRO]sp.if leftvalue=""
rightvalue=[sp_block][sp_macro]sp.erl.message.of.the.day[/sp_macro][
sp_block]
condition="NE"
then=[sp_block]
<a href="[sp_macro]sp.generate_url
nextform="motd.htm"[/sp_macro]">Important messages</a> from the
server administrator
[/sp_block]
[/sp_macro]
```

---

#### sp.admin.total.used

Example: `[SP_MACRO]sp.admin.total.used[/SP_MACRO]`

This macro provides the total amount of charges for records used during a WebSPIRS UBP session. The `sp.admin.initial.used.p` macro variable, which contains the user's initial balance, is used with `sp.admin.total.used`. It is used in calculating the amount used in the session. It is found on the `tbar.htm` template and displayed on the Search page.

---

#### sp.admin.balance

Example: `[SP_MACRO]sp.admin.balance[/SP_MACRO]`

This macro provides the account balance when using the WebSPIRS UBP client. It is found on the `tbar.htm` template and displayed on the Search page.



## Macro Variables

The following are administration macro variables:

---

### sp.erl.server.address

Example: `[SP_MACRO] sp.erl.server.address [/SP_MACRO]`

This macro variable is used to hold the server addresses when iterating through them as part of the Message of the Day routine.

---

### sp.webspirs.docdir

Example: `[SP_MACRO] sp.webspirs.docdir [/SP_MACRO]`

The value of this macro variable is the directory containing the WebSPIRS documentation. It is used in the `foot.htm` fragment to include a hyperlink to the WebSPIRS documentation. It is set in the `webspirs.cfg` file.

```
<A HREF="[SP_MACRO] sp.webspirs.docdir [/SP_MACRO] start.htm">WebSPIRS</A>
```

## General Database Macros

This section describes the macro commands and macro variables which provide the database operations.

### Macro Commands

Following are the general database macro commands:

---

#### sp.avail.dbs.foreach

Example:

```
[SP_MACRO] sp.avail.dbs.foreach
  criteria=sp.avail.dbs.criteria.p
  normalfile="dblitmck.htm"
  titlefile="dblttlck.htm"
[/SP_MACRO]
```

This macro inserts a multiple select list of databases that match the select criteria. It is found on the `database.htm` template. See how the `sp.avail.dbs.foreach` macro looks after the user has logged in, the HTML template has been processed, and it is displayed by your browser as part of the Select Databases form .

---

#### sp.opened.dbs.foreach

Example:

```
[SP_MACRO] sp.opened.dbs.foreach
file="<template_fragment>.htm"
[/SP_MACRO]
```

When a user selects more than one database from the database list, this macro loops through the opened databases and inserts a graphic of a CD before each of the databases being searched. The `file=` parameter contains a template fragment with the database name and titlescreen. This macro is found on the `sel dbs.htm` template. See how a sample list of databases looks when the template has been processed and the list appears on your browser under the toolbar on the Search page.

## Macro Variables

Following are the general database macro variables:

---

### sp.avail.dbs.item.id

Example: `[SP_MACRO]sp.avail.dbs.item.id[/SP_MACRO]`

This macro variable provides the database identifier. It is found on the `dblitmck.htm` template.

---

### sp.avail.dbs.item.name

Example: `[SP_MACRO]sp.avail.dbs.item.name[/SP_MACRO]`

This macro variable provides the displayable name of a database. It is found on the `dblitmck.htm` and `dblttlck.htm` templates.

---

### sp.opened.dbs.item.name

Example: `[SP_MACRO]sp.opened.dbs.item.name[/SP_MACRO]`

This macro variable produces the name of an opened database. It is used in the `dbitem.htm` template fragment.

---

### sp.avail.dbs.item.indent

Example: `[SP_MACRO]sp.avail.dbs.item.indent[/SP_MACRO]`

This macro variable provides the indent prefix which makes an indentation in the structured database list. For HTML purposes, an unordered list `<UL>...</UL>` is used. This variable is used on the `dblitmck.htm` and `dblttlck.htm` templates.

---

### sp.avail.dbs.item.endindent

Example: `[SP_MACRO]sp.avail.dbs.item.endindent[/SP_MACRO]`

This macro variable provides the indent suffix which ends an indentation in the structured database list. For HTML purposes, an unordered list `<UL>...</UL>` is used. It is used on the `dblitmck.htm` and `dblttlck.htm` templates.

---

### sp.dbid.p

Example: `[SP_MACRO]sp.dbid.p[/SP_MACRO]`

This macro variable provides the opened database ID(s). Comma-delimited database IDs will cause the identified databases to be searched as a unit. It is used on the `dbitem.htm` and `dblitmck.htm` templates.

## sp.avail.dbs.criteria.p

Example:

```
[SP_MACRO] sp.avail.dbs.foreach criteria="sp.avail.dbs.criteria.p"
value="P" [/SP_MACRO]
```

This macro variable specifies a set of criteria to be used in opening a database or presenting a list of available databases for selection. Supplying SETID=XX causes any database matching the setid to be included in the list. Any other input will cause the database to be opened temporarily. If it has a thesaurus, the user's input is checked for a match; otherwise, the database's "vocabulary map to fields" are searched. Only databases which match in the thesaurus or the "vocabulary map to fields" may be selected and searched using the value of the criteria for the search. It is found on the `database.htm` template.

## Text Display Macros

This section describes the macro commands and macro variables which are used to display information about records retrieved by searches.

### Macro Commands

Following are the text display macro commands:

---

#### sp.record.toc

Example: `[SP_MACRO] sp.record.toc[/SP_MACRO]`

This macro inserts the table of contents for a full text record. The record number is assumed to be contained in the variable "sp.tocrec". It is found in the `recdtoc.htm` template.

---

#### sp.currentrecord.absolute.url

Example:

```
[SP_MACRO] sp.currentrecord.absolute.url
field="*URLfield"
showtemplate="showlrec.htm"
[/SP_MACRO]
```

This macro is used to generate the URL in the usage-based pricing (UBP) WebSPIRS titles display. The parameter *field*= is used to specify the field whose text is the visible, clickable part of the URL (usually blue), and *showtemplate*= is the template to be used to display the records if the user follows the generated URL. The *field* contains what WebSPIRS thinks is the best field for a \*URL. It is used in the `recfmtck.htm` template.

---

#### sp.currentrecord.abstract.cost

Example: `[SP_MACRO] sp.currentrecord.abstract.cost[/SP_MACRO]`

This macro is used with UBP WebSPIRS. It provides the cost of the current record. It is used in the `recfmtck.htm` template.

## sp.record.text

Example:

```
[SP_MACRO] sp.record.text
filename="recfmtck.htm"
[/SP_MACRO]
```

This macro displays the requested records. The `recfmtck.htm` file provides the format for each record with a marking checkbox. This macro is used on the `show1rec.htm` template.

---

## sp.record.counts

Example:

```
[SP_MACRO] sp.record.counts
source=sp.record.source.p
[/SP_MACRO]
```

This macro displays the number of records yielding hits for the current search. It is used in the `srchcomm.htm` template to display the number of records on the Search page. It is also used on several other templates.

---

## sp.record.initialize

Example: `[SP_MACRO] sp.record.initialize[/SP_MACRO]`

This macro makes sure all scroll values have been calculated and performs any specified searches. It is used on the `rechits.htm`, `recprint.htm`, and `show1rec.htm` templates.

---

## sp.makespurl

Example:

```
[SP_MACRO] sp.checked selections=sp.record.marked.pp
current="sp.makespurl"[/SP_MACRO]
```

This macro build a "SPURL" syntax string to the current record. It is found on the `recfmt.htm` and `recfmtck.htm` templates and used in marking records. The checkbox value is the result of `sp.makespurl` syntax to specify a record. See the description of class `core_SPURL` in the *SilverPlatter CORE Wrapper Reference Manual*.

---

## sp.url.p

Example: `[SP_MACRO] sp.url.p[/SP_MACRO]`

This macro variable contains the "SPURL" syntax hotlink which specifies the next record to be extracted, text, graphics, and portions of records. It is found on the `recsmark.htm`, `mrkclear.htm`, `search.htm`, and `srchcomm.htm` templates.

## Macro Variables

Following are the text display macro variables:

---

### sp.record.fields.p

Example: `[SP_MACRO]sp.record.fields.p[/SP_MACRO]`

This macro variable provides the fields to display in `sp.record.text`. This is used on the `head.htm`, `rechits.htm`, `fielditm.htm`, `recfmtck.htm`, `reprint.htm`, and `showlrec.htm` templates.

---

### sp.record.number.p

Example: `[SP_MACRO]sp.record.number.p[/SP_MACRO]`

The value of this macro variable is the starting record number to display. It is used on the `rechits.htm`, `reprint.htm`, `urlsrch.htm`, and `srchcomm.htm` templates.

---

### sp.currentrecord

Example: `[SP_MACRO]sp.currentrecord[/SP_MACRO]`

This macro variable contains the record number of the currently retrieved record. It is found on the `recfmt.htm` and `recfmtck.htm` and used by the macros that get specific things from the current record.

---

### sp.currentrecord.dbname

Example: `[SP_MACRO]sp.currentrecord.dbname[/SP_MACRO]`

This macro variable contains the database name of the current record. It is found on the `recfmt.htm` and `recfmtck.htm` templates.

---

### sp.record.howmany.p

Example: `[SP_MACRO]sp.record.howmany.p[/SP_MACRO]`

This macro variable provides the number of records to display in `sp.record.text`. It is found on the `head.htm`, `rechits.htm`, `search.htm`, and `scroll.htm` templates.

---

### sp.record.source.p

Example:

```
[SP_MACRO]sp.assign reset="yes" macro="sp.record.source.p"
value= "_HOTLINK"[/SP_MACRO]
```

This macro variable contains "SEARCH" to indicate that a search is to be done or "\_HOTLINK" to indicate that the record(s) are to come from a hotlink (in databases that have hotlinks between records). If it contains SEARCH, it looks for the `sp.search.value.p` variable; if HOTLINK, it looks for the `sp.url.p` variable. It is found on the `rechits.htm`, `recsmark.htm`, `srchcomm.htm`, `hotlink.htm`, and `showlrec.htm` templates.

**sp.record.labels.p**

Example: `[SP_MACRO] sp.record.labels.p[/SP_MACRO]`

This macro variable contains a value that specifies whether long, short, or both field names are to be displayed for the field label options. It is used in the `head.htm`, `reprint.htm`, and `setfield.htm` templates.

**sp.record.lastshown.p**

Example: `[SP_MACRO] sp.record.lastshown.p[/SP_MACRO]`

This macro variable stores the last record number in a list of returned records; for example, if five records are returned, the form will display "1 of 5", and "5" will be the last record. It is used in the `rechits.htm` and `reprint.htm` templates.

**sp.record.sortlimit.p**

Example:

```
[SP_MACRO] sp.isinlist selections=sp.record.sortlimit.p
value="100" iftrue="SELECTED"[/SP_MACRO]
```

This macro variable contains the maximum number of records that can be sorted. If the number of records retrieved in a search exceeds this value, they will not be sorted. It is used in the `setfield.htm` template.

**sp.record.sortfields.p**

Example:

```
[SP_MACRO] sp.field.list sortlist="YES" criteria="*A"
selected=sp.record.sortfields.p[/SP_MACRO]
```

This macro variable provides the fields to sort, such as "PY,D,AU,A." It needs the direction as well as the field to sort. For example "SO,A" sorts the SO field in ascending order. Note that two fields can be supplied if `sp.record.sortfields.p` is multiple select. It is used in the `setfield.htm` template.

**sp.record.sortrecords.p**

Example: `[SP_MACRO] sp.record.sortrecords.p[/SP_MACRO]`

This macro variable has a value of "YES" or "NO". If the value is "YES", sorting will occur. In the following HTML example, macros provide a checkbox for enabling/disabling sorting and a drop down list for choosing the sort arguments:

```
<input type="checkbox" name="sp.record.sortrecords.p"
  [sp_macro]sp.checked selections=sp.record.sortrecords.p
  value="YES">
<SELECT NAME= "sp.record.sortfields.p">
  [SP_MACRO]sp.field.list selected=sp.record.sortfields.p
  sortlist="YES"
  [/SP_MACRO]
</SELECT>
```

## sp.hotlink.form.p

Example:

```
[SP_MACRO]sp.assign macro="sp.hotlink.form.p"
value="hotlink.htm" [/SP_MACRO]
```

This macro variable contains a template to be used when following a hotlink. It is used by `hotlink.htm` and set in `head.htm`.

## Search Macros

This section defines the macro command and macro variables which define the search and search history.

### Macro Command

Following is the search history macro command:

---

## sp.searchhistory.build

Example:

```
[SP_MACRO]sp.searchhistory.build
LABELS= "sp.searchhistory.labels.p"
COUNTS="sp.searchhistory.counts.p"
LIST="sp.searchhistory.list.p" [/SP_MACRO]
```

This macro rebuilds the search history from FORM variables. It should be placed before any macro that will cause the current search to be performed. It adds the current search to the search history. The Search History displays on the Search page. The `sp.searchhistory.labels.p` variable provides identifying information before the search history term, the `sp.searchhistory.counts.p` variable provides the number of hits for the term, and the `sp.searchhistory.list.p` provides the search history list. It is used in the `search.htm` template.

### Macro Variables

Following are the search and search history macro variables:

---

## sp.search.value.p

Example: `[SP_MACRO]sp.search_value.p[/SP_MACRO]`

This macro variable supplies the search in SilverPlatter Information Retrieval System (SPIRS) **find parser** format. It is used in the `clsrch.htm`, `database.htm`, `search.htm`, `scroll.htm`, and `srchcomm.htm` templates.

---

## sp.searchhistory.operator

Example: `[SP_MACRO]sp.searchhistory.operator[/SP_MACRO]`

This macro variable contains the Boolean operator (AND, OR, NOT, or REMOVE) to be used in combining terms selected from the search history list. It is used in the `srchcomm.htm` template.

## Automatic Subject Lookup Macros

The following macro is used for automatic subject lookup (ASL):

---

### sp.asl.list

Example:

```
[SP_MACRO] sp.asl.list
termlist="sp.asl.terms"
phrase=sp.asl.phrase.p
selected=sp.select.terms.pp
limit="10"
[/SP_MACRO]
```

This macro provides a list of the ASL terms selected by the user. It is used on the `suggest.htm` template. The user clicks the Suggest button on the toolbar, enters a term, and a list of suggested terms and term details displays. Note that the Suggest button appears only when specific databases are selected.

Parameters:

<i>termlist</i>	Provides the list of terms.
<i>phrase</i>	Specifies the word or phrase the user is interested in.
<i>selected</i>	Any previously selected terms.
<i>limit</i>	How many terms the ASL is to match.

## Field-Specific Index Macros

This section describes the macro commands and macro variables which are used with index searches.

### Macro Commands

Following are the field-specific index macro commands:

---

### sp.fsi.tosearch

Example:

```
[SP_MACRO] sp.fsi.tosearch
macro="sp.search.value.p"
reset="yes"
terms=sp.fsi.term.p
fields=sp.fsi.fields.p
[/SP_MACRO]
```

This macro takes an ORed search of terms and fields and puts the result in `sp.search.value.p`. It is used on the `sugsrch.htm` and `srchidx.htm` templates.

Parameters:

<i>macro</i>	Specifies the variable to hold the search.
<i>reset</i>	Specifies whether the new search is to be concatenated with the OR operator. If the value is NO or omitted, it OR's the ORed list of terms onto the search. If the value is YES, the list of terms becomes the search.
<i>terms</i>	A comma-delimited list of terms.
<i>fields</i>	The fields in which to search.



## sp.fsi.list

Example:

```
[SP_MACRO]sp.fsi.list termlist="sp.fsi.terms"
recordslist="sp.fsi.counts" [/SP_MACRO]
```

This macro provides

on the `index.htm` template. When the user clicks the Index button on the toolbar, then enters a term, a list of terms displays.

---

## sp.fsi.copytovvariable

Example:

```
[SP_MACRO]sp.fsi.copytovvariable
MACRO="sp.limit.la.contents"
FSI="LA"
longnamecontains="language" [/SP_MACRO]
```

This macro copies the contents of a field-specific index (FSI) to a specified variable. It allows FSIs to be turned into Select lists. In the above example, the list "ALBANIAN, FRENCH, IROQUOIS" might be put in the variable "sp.limit.la.contents." It is used in the `srchcomn.htm` template to create the limit lists.

### Parameters:

<i>macro</i>	The variable name to contain the comma-delimited list of macro variables.
<i>fsi</i>	The index to be used to supply the values.
<i>longnamecontains</i>	The long name of the value in <i>fsi</i> .

## Macro Variables

Following are the field-specific index macro variables:

---

## sp.fsi.fields.p

Example: `[SP_MACRO]sp.fsi.fields.p[/SP_MACRO]`

The `sp.fsi.fields.p` variable is used to present a list of fields to the user on the Index page.

The following section from `index.htm` shows how this macro variable is used:

```
<H2>Index of field "[SP_MACRO]sp.fsi.fields.p[/SP_MACRO]"</H2>
<p>Type all or part of a term, and select the index you would like to see, then press
  Display.<br>
  Click on a term to search it.</p>
<p><b>Term:</b> <INPUT NAME="sp.fsi.term.p" SIZE="50"><br>
<B>Field:</B>
<SELECT NAME="sp.fsi.fields.p">
<option value="*F" [sp_macro]sp.isinlist selections=sp.fsi.fields.p value="*F"
  iftrue="SELECTED"[/sp_macro]>Free Text</option>
[SP_MACRO]sp.field.list criteria="*Separate" selected=sp.fsi.fields.p[/SP_MACRO]
</SELECT>
```

## sp.fsi.term.p

Example:

```
[SP_MACRO]sp.assign macro="sp.fsi.term.p"
value="" [/SP_MACRO]
```

This macro variable provides the term the user wants to look up in the dictionary. It is used on the `index.htm` template.

---

## sp.fsi.howmany.p

Example: `[SP_MACRO]sp.fsi.howmany.p[/SP_MACRO]`

This macro variable provides the value of how many items to scroll up or down. It is used in the `index.htm` template to create a dropdown list as follows:

```
<A href="#howmanydrop">[sp_macro]sp.fsi.howmany.p[/sp_macro]
terms</a>
```

## Field List Macro

The following macro command is used when working with a list of fields:

---

## sp.field.list

Example:

```
[SP_MACRO]sp.field.list
criteria="*SEPARATE"
selected=SP.FSI.FIELDS.P
[/SP_MACRO]
```

This macro offers a select list of fields; uses `sp.field.criteria` to choose, and puts the selection in `sp.field.choice`. It is used in the `fieldditm.htm`, `index.htm`, and `setfield.htm` templates, and it takes arguments:

*Parameters:*

*criteria*                Specifies the fieldset (as a comma-delimited list of fields) to be included in the list.  
*selected*                A comma-delimited list of fields already selected by the user.

The list of available fields appears when displayed by your browser after the words, "Fields to Show:" on the Field Options list page. This page is accessed by the "Fields" and "Custom Fields..." buttons.

## Thesaurus Term Macros

The macros in this section are used to display information about a thesaurus term and to allow the user to pick, narrower, broader, and related terms to be used in searching. When the user clicks the Thesaurus button on the toolbar and then enters a term, a Thesaurus Subject List (permuted list) is displayed. After selecting a term from the list, the Thesaurus Term page displays with lists of selectable broader (more general) terms and topical subheadings for the selected term.

---

## Macro Commands

Following are the thesaurus term macro commands:

---

### sp.term.definition

Example:

```
[SP_MACRO] sp.term.definition
term=sp.thesaurus.term.p
[/SP_MACRO]
```

This macro inserts the definition of the term specified by *term*. It is used in the `sugtermd.htm` and `theterm.htm` templates.

---

### sp.term.prepare\_detail

Example:

```
[SP_MACRO] sp.term.prepare_detail
term=sp.thesaurus.term.p
[/SP_MACRO]
```

This macro creates a number of internal comma-delimited lists for usage by the `sp.foreach` macro. It is used in the `theterm.htm` template.

#### Parameters:

*ter*            The thesaurus term to be prepared.  
*m*

Variables created are:

- `sp.term.related_terms`--A comma-delimited list of terms related to this one.
  - `sp.term.narrower_terms`--A comma-delimited list of terms with narrower meaning to this one.
  - `sp.term.broader_terms`--A comma-delimited list of terms with narrower meaning to this one.
  - `sp.term.search.p`--Search to be used to find records for this term.
- 

### sp.expansionpart

Example:

```
[SP_MACRO] sp.expansionpart
text=sp.check.value
[/SP_MACRO]
```

This macro is used in the `checkbox.htm` template. Subheadings (stored in the `sp.age.subheading` variable) are entered in the format `searchvalue@readablevalue`. This macro splits off the readable value so it can be displayed to the user.

## Macro Variables

Following are the thesaurus term macro variables:

---

### sp.select.terms.pp

Example: `[SP_MACRO] sp.select.terms.pp [/SP_MACRO]`

This is a macro variable used in the `suggest.htm` and `theterm.htm` templates. The `sp.foreach` macro assigns it the values of each of a comma-delimited string of terms. See the `sp.foreach` macro for additional information.

---

### sp.perm.word.p

Example:

```
[SP_MACRO] sp.term.permuted_list
termfrom="theterm.htm"
word=sp.perm.word.p [/SP_MACRO]
```

This macro variable is the word being looked up in the permuted list. It is used in the `subject.htm` template.

---

### sp.age.subheading

Example: `[SP_MACRO] sp.age.subheading [/SP_MACRO]`

This macro variable contains a comma-delimited list of age subheadings the user has chosen to search.

---

### sp.topical.subheading

Example: `[SP_MACRO] sp.topical.subheading [/SP_MACRO]`

This macro variable contains a comma-delimited list of topical subheadings the user has chosen to search. It is used in the `srchterm.htm` template.

---

### sp.term.narrower\_terms

Example:

```
[SP_MACRO] sp.foreach iterator="LIST"
list=sp.term.narrower_terms
variable="sp.select.term"
filename="thesitem.htm" [/SP_MACRO]
```

This temporary macro variable contains a term's narrower (more specific) terms.

---

### sp.term.related\_terms

Example:

```
[SP_MACRO] sp.foreach
iterator="LIST" list=sp.term.related_terms
variable="sp.select.term"
filename="thesitem.htm" [/SP_MACRO]
```

This temporary macro variable contains a term's related terms. It is used in the `theterm.htm` template.

## **sp.thesaurus.term.p**

Example: `[SP_MACRO] sp.thesaurus.term.p[/SP_MACRO]`

This macro variable contains the current thesaurus term. It is used in the `sugterm.htm`, `sugtermd.htm`, `thesitem.htm`, and `thesterm.htm` templates.

## **Guide Keyword Macros**

This section describes the macro commands and macro variable which are used to access the database help (guides) in a hypertext form.

### **Macro Commands**

Following are the guide macro commands:

---

## **sp.guide.toc**

Example: `[SP_MACRO] sp.guide.toc[/SP_MACRO]`

This macro builds the guide tables of contents. It is used in the `guidtoc.htm` template.

---

## **sp.guide.topic**

Example:

```
[SP_MACRO] sp.guide.topic
topic=sp.guide.topic
type=sp.guide.topic_type
[/SP_MACRO]
```

This macro provides topics for a keyword chosen by `sp.guide.keyword`. It is used in the `guidtopic.htm` template. The user selects a topic from the guide table of contents.

### **Macro Variable**

Following is the guide macro variable:

---

## **sp.guide.dbname**

Example:

```
[SP_MACRO] sp.assign macro="sp.template_description"
interpret="yes" value="sp.guide.dbname{-Guides}"[/SP_MACRO]
```

This macro variable is used in the `guidtopic.htm` template. It provides the database name for the guide topic.

## Marked Records

Users can mark search results records and then display the list of marked records. Several macros are involved in this process, including the macros which create checkboxes. Checkboxes are also used for other purposes, such as selecting databases. This section describes the macro commands and macro variables used in the marking of records.

## Macro Commands

Following are the macro commands used in the marking of records:

---

### sp.checked

Example:

```
[SP_MACRO] sp.checked
  selection=sp.record.marked.pp
  value=sp.makespurl
  current= "sp.makespurl"
[/SP_MACRO]
```

This macro is used in generating a checkbox (`<INPUT type="checkbox">`). It fills in the word checked by the user if the *value=* part is contained in the *selection=* part. The *current=* parameter is a macro to run to get the current value. This macro is used in the `recfmt.htm`, `recfmtck.htm`, and `checkbox.htm` templates.

---

### sp.isinlist

Example:

```
[SP_MACRO] sp.isinlist
  selections=sp.limit.use.p
  value="1"
  iftrue="CHECKED"
[/SP_MACRO]
```

This macro supercedes the `sp.checked` macro. The parameter *selections=* supplies the list of selected values, *value=* is the current value, and *iftrue=* provides the text to be supplied if the value is in the selections; for example, "CHECKED". It is used in the `sugterm.htm` and `index.htm` templates.

## Macro Variables

Following are the macro variables used in the marking of records:

---

### sp.check.name

Example: `[SP_MACRO] sp.check.name[/SP_MACRO]`

This temporary macro variable is used in generating a checkbox. It contains the variable name the checkbox is to use; that is, the value of the HTML `NAME=`. It is used in the `checkbox.htm` template.

## sp.check.selections

Example:

```
[SP_MACRO]sp.checked selections=sp.check.selections
value=sp.check.value[/SP_MACRO]
```

This temporary variable is used by the `sp.checked` macro to check items previously selected by the user. It has a comma-delimited list of the selected values which is compared to the current value of each checkbox. If there is a match, the checkbox is checked. It is used in the `checkbox.htm` template.

---

## sp.record.marked.pp

Example: `[SP_MACRO]sp.record.marked.pp[/SP_MACRO]`

This macro variable holds the list of records marked by the user. It is used in the `recsmark.htm`, `mrkclear.htm`, `recfmt.htm`, and `recfmtck.htm` templates. The `.pp` extension causes it to be handled somewhat differently from the `.p` extension. When using multiple values that may or may not be displayed together, the `.pp` extension writes out the variable if it doesn't find the value in one of the `<INPUT>` statements. (The `.p` extension writes out the variable as long as there is no `<INPUT>` or `<SELECTED>` statement for the name.)

## Miscellaneous Macro Support

Other macros are also included which do not fit into the above categories. This section describes the miscellaneous macro commands and macro variables.

## Macro Commands

Following are the miscellaneous macro commands:

---

## sp.assign

Example:

```
[SP_MACRO]sp.assign
macro="sp.search.value.p"
reset="yes"
interpret="yes"
value="{() sp.search.term1.p {} }
      {() sp.search.type2.p { () sp.search.term2.p {} } }
      {() sp.search.type3.p { () sp.search.term3.p {} } }"
[/SP_MACRO]
```

The `sp.assign` macro is used to set variables. This macro is found in most templates, and the `head.htm` template has many examples of its use. It places the contents of the *value* parameter in the variable specified by the *macro* parameter.

### Parameters:

<i>macro</i>	Specifies the macro variable whose value is to be assigned.
<i>reset</i>	This equals either "yes" or "no". If "yes", the assignment takes place whether or not the macro currently has value. If "no", the assignment only takes place if the variable has no value. This is used to establish default values. If not specified, "reset=no" is assumed.
<i>interpret</i>	This equals either "yes" or "no". If "yes", the value is interpreted based on rules specified in the value parameter; if "no", the contents of <i>value</i> = is treated as a literal. If not specified, "interpret=no" is assumed.
<i>value</i>	The value that is to be assigned to the macro variable. Rules for interpreting the value allow complex strings built from both literals and other macro variables to be built up.

Following are the rules for the `sp.assign` macro:

- Literal strings must be enclosed by braces (`{ }`).
- Macro variables are left unquoted. Their values are inserted in the string.
- In clauses surrounded by `(...)` all the variables specified must be present.

#### Usage examples:

In the above `sp.assign` example, the value shown is placed in the `sp.search.value.p` macro. In the following usage examples, the macro assigns "cat" or "dog."

```
sp.search.term1.p = cat sp.search.type2.p = or sp.search.term2.p = dog
sp.search.type3.p = or sp.search.term3.p = mouse value=" {({}
sp.search.term1.p {)} ( { } sp.search.type2.p {({} sp.search.term2.p {)}} (
{ } sp.search.type3.p {({} sp.search.term3.p {)}} evaluates to: "(cat) or
(dog) or (mouse)" 2. sp.search.term1.p = cat sp.search.type2.p = or
sp.search.term2.p = dog sp.search.type3.p = or sp.search.term3.p =
evaluates to: "(cat) or (dog)" (term3 is empty so there is no mouse.)
```

---

## sp.foreach

Example:

```
[SP_MACRO] sp.foreach
iterator="SETID"
variable="sp.guide.dbid.p"
filename="guidprdb.htm"
[/SP_MACRO]
```

This macro repeatedly expands the macros contained in the specified filename. The expansion is controlled by the *iterator* type.

#### Parameters:

- iterator* Specifies the kinds of iterations. Currently there are three kinds: SETID, DATABASE, and LIST. SETID causes the macros contained in the specified filename to be expanded once for each unique database SETID in the databases that are being searched. The DBID of one of these is placed in the variable specified by the variable parameter. This allows things such as guides and titles to be included only once. DATABASE causes the macros contained in the specified filename to be expanded once for each database. The DBID is placed in the variable specified by the *variable* parameter. LIST causes the macros contained in the specified filename to be expanded once for each value in a comma-delimited list specified by the additional *list* parameter.
- list* This is a comma-delimited list only valid for the LIST iterator.
- variable* This is the "macro variable" where the iterated value is to be placed. The macros being expanded (read in from the specified file) can make use of this; for example, it can be used to open a specific guide database.
- filename* Any file containing text and or macros to be repeated or inserted into the template.



## sp.generate\_url

Example:

```
[SP_MACRO] sp.generate_url
    nextform="theterm.htm"
    args="sp.thesaurus.term.p=sp.select.term"
[/SP_MACRO]
```

This macro generates a URL to another template within WebSPIRS. It allows another means of navigation beyond buttons.

### Parameters:

*nextform* Specifies the template to be loaded if the URL is followed.

*args* Specifies arguments to be added onto the URL. The "usernumber" and the "sp.dbid.p" are added by default. *args* are evaluated so that the macro values to the right of the '=' are replaced with their values. Multiples of "macro=value" must be strung together with "&". In the above example, the URL might be:

```
"http://.../.../webspirs?sp.nextform=theterm.htm&sp.usernumber.p=4&sp.dbid.p=MLSB&sp.thesaurus.term.p=neoplasms"
```

---

## sp.include

Example:

```
[SP_MACRO] sp.include
    filename="srcrrslt.htm"
    source='_search'
    record='1'
    action='record.show'"
[/SP_MACRO]
```

The `sp.include` macro reads in the template file specified by `filename=`, interprets any macros, and inserts the result in the form. Text after the filename is treated as arguments.

---

## sp.if

Example:

```
[SP_MACRO] sp.if condition="EQ"
leftvalue="1"
rightvalue=sp.search.numberofinputs.p
then=[sp_block]...[/sp_block]
else=[sp_block]...[/sp_block]
[/SP_MACRO]
```

The `sp.if` macro provides for a conditional "if-else" programming statement. In the above example, if the `leftvalue` (1) equals the `rightvalue` (the `sp.search.numberofinputs.p` variable), then an action is performed. Or `else` (if the left and right values are not equal), then another action is performed. Note that the `else` parameter is optional.

This macro evaluates a condition. The `condition=` value can be `EQ` (equal), `NE` (not equal), `GT` (greater than), `LT` (less than), `GE` (greater than or equal to), or `LE` (less than or equal to). `EQ` will work if you are comparing numbers or strings. If you don't set anything, then you are comparing strings. If there is a `type=` parameter and it is "numeric", the `leftvalue` and the `rightvalue` must be numbers. Each of the two resulting `[sp_block] ...[/sp_block]` set of tags contain macro-encoded HTML.

Variations of the `sp.if` macro are the `sp.iffirst`, `sp.iflast`, `sp.ifnotfirst`, and `sp.ifnotlast` macros.

---

## Macro Variables

Following are the miscellaneous macro variables:

---

### sp.output

Example:

```
[SP_MACRO] sp.assign
macro="sp.output"
value= "mail" [/SP_MACRO]
```

The `sp.output` macro variable is used with the "", "mail", and "file" values. It sets the next page to the screen, mail, or a file respectively. It is used in the `exphtml.htm`, `expmail.htm`, and `saveopts.htm` templates.

---

### sp.mailcmd

Example:

```
[SP_MACRO] sp.if leftvalue= "1"
condition= "EQ"
rightvalue= sp.mailcmd
then=[sp_block].....[/sp_block] [SP_MACRO]
```

This macro variable is set in the `webspirs.cfg` file to a value of "1" and indicates that mail is activated. It runs the `spsmail` script when mail is selected. When WebSPIRS runs the command for the script, it invokes two parameters. The first is the filename of the text to mail, and the second is the Email address the user fills in on the Mail Records page. This variable is used on the `rechits.htm` template.

---

## Adding Macros

If you want to create additional macros, you can do so using the C++ programming language. You will also need the following:

- C++ compiler to add functions
- Text editor to test and configure the interface
- Code libraries and related documentation

For documentation you will need the *SilverPlatter CORE Reference Manual* and the *SilverPlatter CORE Wrapper Reference Manual*. The details of the WebSPIRS application program interface (API) are provided in Chapter 5.

If you are planning to create new macros, send an electronic mail message to Yogen Pathak at [YogenP@SilverPlatter.com](mailto:YogenP@SilverPlatter.com) and request to become a participant in the WebSPIRS Developer Program. He can also provide copies of the appropriate SilverPlatter documentation.

Helpful CGI test scripts and form decoding software programs can be found in the Common Gateway Interface specification .

## Building WebSPIRS

You can build WebSPIRS on either a 32-bit Windows operating system or on a UNIX system. The following steps explain how to build WebSPIRS on a UNIX system:

1. Unzip the `webspirs.zip` file. The files will unpack into the following directories:

- `cgibaby`
- `cgichild`
- `cgiadult`
- `include`
- `cgiprocs`

Note: The type of HTTP server you use can influence the directory structure.

2. Use the make utility:

```
make
```

3. Run WebSPIRS by doing the following:

```
cgichild&
```

```
(sends test.req request)
```

4. Edit the `erlclnt.cfg` file to include the server information.

5. Edit the `webspirs.cfg` file and change the "template\_directory"

Instructions for editing the configuration files can be found in Chapter 2, "Installing and Configuring WebSPIRS."

## C++ Language Implementation

The C++ macros are slightly more complicated to set up initially but are easier to implement than the C macros, and they offer a means (through the derived `www_HTML_Helper` class) to allocate things that persist beyond the scope of the macro. For instance, the `www_Search` class holds the current search and the `www_Database` class holds the current database.

## Implementing a new `www_HTML_Helper` Class

Complete the following steps to implement a new `www_HTML_Helper` class:

1. Code the class in the header file (.hpp). There must be a constructor that takes one argument, a pointer to the `www_Request` class, and there must be a destructor. Here is an example from the `wwwsrch.hpp` header file:

```
class www_Search : public www_HTML_Helper
{
    /*
    purpose:
    <1>hangs onto the search for the length of the request
    <2>interprets the ASL macro
    <3>gets the parsed search text or reports an error
    <4>does thesaurus stuff
    */
public:
    www_Search(www_Request *callersRequest);
    virtual ~www_Search();
    void Reset(); //method called at end of request, we delete the search
    core_RecordArray *GetRecords(const prString &callersSource);
    static const prString &GetASLList(const char *callersMacro,
                                     www_HTML_Helper *callersThis);
    static const prString &GetSearchText(const char *callersMacro,
                                         www_HTML_Helper *callersThis);
    static const prString &GetTermDefinition(const char *callersMacro,
                                             www_HTML_Helper *callersThis);
    static const prString &PrepareTerm(const char *callersMacro,
                                       www_HTML_Helper *callersThis);
private:
    core_Search *mySearch; // record array returned by GetRecords,
                          // class deletes it at the end
};
```

2. In the implementation (.cpp file), declare an instance of the template:

```
"www_HelperConstructor<...>"
```

This serves to connect the newly implemented helper class to WebSPIRS.

Here is an example of a static instance that adds the `www_Search` class constructor to the table of helpers:

```
"www_HelperConstructor<www_Search>MakeAwwwSearch;
```

3. Create a table of macro names and macro expansion functions (much like in the C macro section).

If the functions need to make use of private data held by the "helper" class, they should be static members (as above). Otherwise, they may just be functions with file scope (declared statically within the module).

The last entry in the table should be {0,0} to let the initialization code in `www_HTML_Helper` know that the last macro has been encountered.

Here is an example:

```
static www_HTML_Helper::wwwMacroDef wwwSearchMacroTable[] =
{
    {"sp.asl.list",www_Search::GetASLList},
    {"SP.RECORD.SEARCH",www_Search::GetSearchText},
    {"sp.term.definition",www_Search::GetTermDefinition},
    {"sp.term.prepare_detail",www_Search::PrepareTerm},
    {0,0}
};
```

This table should be passed to the helper via the "Init" method in the constructor. Here's the **www\_Search** constructor:

```
www_Search::www_Search(www_Request *callersRequest)
    :www_HTML_Helper(callersRequest)
{
    mySearch = 0;
    Init(wwwSearchMacroTable);
}
```

## Coding C++ Macro Functions

Complete the following steps to code a C++ macro function:

1. Add the function name to the above table macros.
2. Be sure the macro function prototype looks like the following:

```
FunctionName(const char *callersMacro, www_HTML_Helper
*callersThis);
```

*callersMacro* is the text of the macro starting with the macro itself and ending after its last argument. For example, the **GetASLList** macro might have been coded in the template as:

```
[sp_macro]sp.asl.list
    phrase="happy to meet you"
    limit="5"
[/sp_macro]
```

the string in *callersMacro* would be:

```
sp.asl.list phrase="happy to meet you" limit="5"
```

*callersThis* is the pointer to the **www\_HTML\_Helper** class that contains this macro. It must be cast if you need to use any data special to it. For instance:

```
www_Search *me = (www_Search *)callersThis;
```

3. The macro must return a reference to a string which contains the macro-expanded text. The string may be empty. To get the string filled use:

```
prString &myExpansion = callersThis->GetExpansionString();
myExpansion = prNullString;
```

**Note:** You should make sure to initialize the expansion string as it may contain the value from the last macro the helper expanded.

To add a macro to write "Hello World" to `www_Search` you would:

```
...
...
// function prototype
static const prString &CppHelloWorld(const char *callersMacro,
                                     const char *callersThis);
...
...
// add to function table
static www_HTML_Helper::wwwMacroDef wwwSearchMacroTable[] =
{
    {"sp.asl.list",www_Search::GetASLList},
    {"SP.RECORD.SEARCH",www_Search::GetSearchText},
    {"sp.term.definition",www_Search::GetTermDefinition},
    {"sp.term.prepare_detail",www_Search::PrepareTerm},
    {"sp.cpp.Hello.World",CppHelloWords}
    {0,0}
};
...
...
//
// expand to hello world when [sp_macro]sp.cpp.hello.world[/sp_macro]
// is found in the template
//
const prString &CppHelloWorld(const char *callersMacro,
                              const char *callersThis)
{
    prString &theExpansion = callersThis->GetExpansionString();
    theExpansion = "Hello World";
    return theExpansion;
}
```

## Useful Services Available to the Macro

The following are some generally useful services available to the macro:

- The `www_Request` class is accessible by way of the following:

```
www_Request *request = callersThis->GetRequest();
```

The request contains all the HTTP request values received from the user. They may be accessed via the `core_Config` API (see Chapter 4). Here is an example:

```
prString mySearch =
    callersThis->GetRequest()->GetItem("sp.search.value.p","");
```

The macro is free to store new items in the request as in:

```
callersThis->GetRequest()->StoreItem("sp.newvariable","a new value");
```

If the variable name ends in ".p" the variable will be included in the form as a hidden variable.

- The database(s) involved in the request.

```
core_Database *db = callersThis->GetRequest()->GetDatabase();
if (db == 0)
    return(prNullString);
```

This will open the database if this is the first macro to call `GetDatabase()`.

- Get the set of records currently in use by the request:

```
core_RecordArray *records = callersThis->GetRequest()->GetRecords();
```

Interpretation of the macro arguments is handled by way of the **InterpretArguments** method. If the *callersMacro* contains:

```
... arg1="halleluia" arg2="im a bum"
   core_Config * cfg = callersThis->InterpretArguments(callersMacro);
   myArg1 = cfg->GetItem("arg1","");
   myArg2 = cfg->GetItem("arg2","");
   delete cfg;
```

*myArg1* *myArg2* will contain "im a bum".

- Read a file from the WebSPIRS default template directory:

```
prString LoadGatewayFile(const prString &callersFilename,
                        prString &callersReturnFileName,
                        prString &callersReturnTemplateArguments);
```

The first string is the filename to be loaded, the second string will contain the actual filename loaded, and the third will contain any decorations added at the end of the filename (only used by the **www\_Template** class at the moment). The string returned is the contents of the file.

- The following methods generate various kinds of HTML:

```
//
// makes <option value="callersValueArg">callersText</option> entry
// will insert SELECTED attribute if value= matches contents of
// comma-delimited list in callersSelections
//
prString MakeOptionHTML(const char *callersValueArg,
                       const char *callersText,
                       const prString &callersSelections);

//
// makes a checkbox: <INPUT TYPE="CHECKBOX"
// NAME="callersVariableName"
// VALUE="callersValue">callersText</input>
// check box is CHECKED if value matches one of the comma-delimited
// values in callersSelections
//
prString MakeCheckboxHTML(const char *callersVariableName,
                          const char *callersValue,
                          const char *callersText,
                          const prString &callersSelections);

//
// makes the <a href="">prompt</a> markup
//
prString MakeHyperLink(const prString &callersPrompt,
                       const prString &callersSHREF);

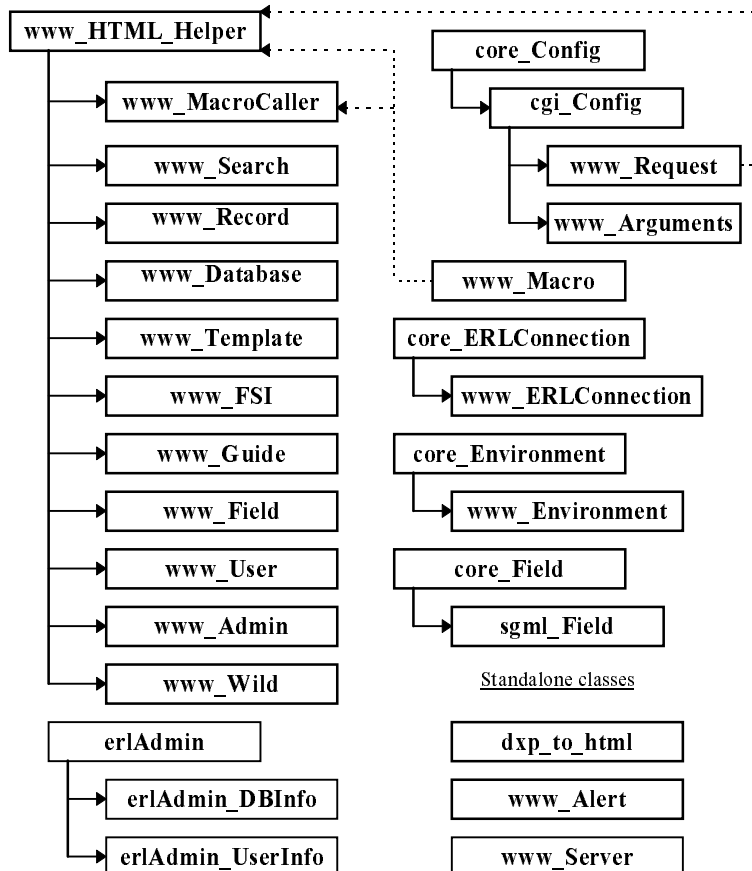
//
// creates a URL to the gateway aimed at the specific form
// this will include certain defined bits of context:
//   sp.usernumber.p & sp.dbid.p, anything else can be shoved in
//   in extra arguments as tag=value&tag=value&...
// if the DbId argument is left prNullString, the sp.dbid.p= will default to the
//   currently open database
//
prString MakeUrl(const prString &callersDestinationForm,
                const prString &callersDbId = prNullString,
                const prString &callersExtraArguments = prNullString);
```

# Chapter 5 - WebSPIRS Class Library

This chapter details the WebSPIRS application program interface (API) which was written in the C++ programming language and includes the following classes:

- **cgi\_Config**
- **dxp\_to\_html**
- **erlAdmin**
- **erlAdmin\_DBInfo**
- **erlAdmin\_UserInfo**
- **sgml\_Field**
- **www\_Admin**
- **www\_Alert**
- **www\_Arguments**
- **www\_Database**
- **www\_Environment**
- **www\_ERLConnection**
- **www\_Field**
- **www\_FSI**
- **www\_Guide**
- **www\_HTML\_Helper**
- **www\_Macro**
- **www\_MacroCaller**
- **www\_Record**
- **www\_Request**
- **www\_Search**
- **www\_Server**
- **www\_Template**
- **www\_User**
- **www\_Wild**

The classes appear in this chapter in alphabetical order. The following hierarchical class drawing shows the inheritance and relationships of the classes. In the drawing, a solid line indicates inheritance and a broken line indicates relationship. The classes **core\_Config**, **core\_Environment**, **core\_ERLConnection**, and **core\_Field**, which are shown in the drawing, are described in the *SilverPlatter CORE Wrapper Reference Manual*.

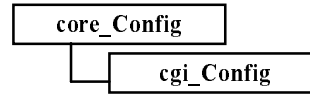




## Class `cgi_Config`

See Also: `cgiconfig.hpp`

The class `cgi_Config` adds HTTP/URL coding and decoding. It handles the reading of common gateway interface (CGI) requests. This class inherits from class `core_Config`.



See the *SilverPlatter CORE Wrapper Reference Manual* for details on the `core_Config` class.

## Public Member Functions

The following are public member functions of class `cgi_Config`:

---

### `cgi_Config`

`cgi_Config();`

A class constructor.

`cgi_Config(const cgi_Config &from);`

A copy constructor.

---

### `~cgi_Config`

`virtual ~cgi_Config();`

The class destructor.

---

### `operator=`

`const cgi_Config &operator = ( const cgi_Config &from);`

The assignment operator.

---

### AddEnvironment

`void AddEnvironment (const char *callersEnvp[]);`

Stores the application's environment `tag=values` in the CORE's **configuration** object.

---

### AddArguments

`void AddArguments (int callersArgc, const char *callersArgv[]);`

Stores the application's arguments in the CORE's configuration object.

---

### AddHttpTags

`void AddHttpTags (const char *callersHttpString);`

Stores the string of URL-encoded `tag=values` in the CORE's **configuration** object. These are not URL encoded.

## AddTags

**void AddTags (const char \*string, char delim = ',');**

Stores a string of comma-delimited values in the CORE's **configuration** object. These are not URL encoded.

---

## RoleInDefaults

**void RoleInDefaults();**

Roles any [ default . ] tags into the real value.

---

## ConvertToHttp

**prString ConvertToHttp();**

Takes all the configuration tags and builds a large URL-encoded string of tag=values.

---

## AppendCfg

**void AppendCfg (core\_Config &callersConfig, boolean callersReplace = cFALSE);**

Takes all the tag=values in *callersConfig* and merges them into itself (**cgi\_Config** class). The *callersReplace* parameter causes existing tags to be replaced. If cTRUE, the tags duplicated will be replaced; if cFALSE, the old values will hold.

---

## MergeItem

**void MergeItem(const char \*callersTag, const char \*callersValue);**

Puts the value in *callersValue* on the end of the value already stored in *callersTag*.

---

## Dump

**prString Dump();**

This is a debug routine to format all the tag=value contents of the configuration.

---

## ReadRequest

**void ReadRequest();**

Reads a request from **stdin** as part of the CGI interface.

---

## URLEncodeString

**static void URLEncodeString (prString &dest, const uchar \*src);**

URL encodes *src* and appends it onto *dest*.

---

## URLDecodeString

```
static void URLDecodeString (prString &dest, const uchar *src);
```

URL decodes *src* and appends it onto *dest*.

---

## RestoreCodedCharacters

```
static prString RestoreCodedCharacters(const char *callersCodedString);
```

This method restores certain coded characters that get in the way of URL Processing. For example, ">" which is used in the find syntax and is turned into .GT.

## Class `dxp_to_html`

**See Also:** File `dxp2htm.hpp`

The standalone class `dxp_to_html` knowledge of how to turn DXP-formatted text into HTML-formatted text.

### Public Member Functions

The following are public member functions of class `dxp_to_html`:

---

#### `dxp_to_html`

`dxp_to_html (core_Database *callersDb =0);`

The class constructor.

---

#### `~dxp_to_html`

`~dxp_to_html()`

The class destructor.

---

#### `SetDatabase`

`dxp_to_html &SetDatabase(core_Database *callersDb);`

This method lets the DXP to HTML converter make use of database information, particularly to build hotlinks.

---

#### `SetTranslateFromDos`

`dxp_to_html &SetTranslateFromDos (boolean callersTranslateFromDos);`

Translates the PC character set to the ISO Latin character set.

---

#### `SetRequest`

`dxp_to_html &SetRequest (www_Request *callersRequest);`

Allows the class to access information about the current request. Used when building a URL for hotlinks.

---

#### `Convert`

`const prString &Convert (const prString &theDXPText);`

Translates DXP text into HTML.

## Class **erlAdmin**

**See Also:** File `erladmin.hpp`

The base class **erlAdmin** establishes congruence between the Network CORE-based network access and the administrative-based network access. It involves the copying of globals. This class should be called after the `core_ERLConnection` class has been constructed.

### Protected Member Functions

The following are the protected member functions of class **erlAdmin**:

---

#### **erlAdmin**

**erlAdmin();**

The class constructor.

---

#### **~erlAdmin**

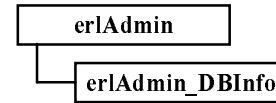
**virtual ~erlAdmin();**

The class destructor.

## Class `erlAdmin_DBInfo`

**See Also:** File `erladmin.hpp`

The helper class `erlAdmin_DBInfo` provides access to administrative information about a server that is not available through the Network CORE, in particular, the cost per record. Information is cached inside the database so that the number of messages is kept to a minimum.



The `SetDatabase` method must be called prior to accessing information because this provides the connection to the database and the place to cache the information.

### Public Member Functions

Following are the public member functions of class `erlAdmin_DBInfo`:

---

#### `erlAdmin_DBInfo`

```
erlAdmin_DBInfo();
```

The class constructor.

---

#### `~erlAdmin_DBInfo`

```
~erlAdmin_DBInfo();
```

The class destructor.

---

#### `GetCostPerRecord`

```
ulong GetCostPerRecord();
```

Returns the cost per record from ERL to WebSPIRS.

---

#### `GetCostPerAbstract`

```
ulong GetCostPerAbstract();
```

Returns the cost per abstract (the AB field) from ERL to WebSPIRS.

---

#### `SetDatabase`

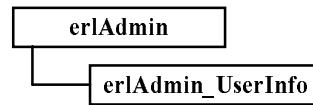
```
erlAdmin_DBInfo &SetDatabase(core_Database *callersDb);
```

Sets the database from which the records will be retrieved. This must be called prior to accessing information. It provides the connection to the database and the place to cache the information. It also identifies the server.

## Class `erlAdmin_UserInfo`

See Also: File

The class `erlAdmin_UserInfo` provides access to administrative information about a user. The information is not cached. The `SetUserName` method must be called prior to accessing the information.



### Public Member Functions

Following are the public member functions of class `erlAdmin_UserInfo`:

---

#### `erlAdmin_UserInfo`

```
erlAdmin_UserInfo();
```

The class constructor.

---

#### `~erlAdmin_UserInfo`

```
virtual ~erlAdmin_UserInfo();
```

The class destructor.

---

#### `SetUserName`

```
void SetUserName(const char *callersUserName);
```

This function sets the username to be inquired about. It must be called prior to accessing information.

---

#### `SetServer`

```
void SetServer(const char *callersServerName);
```

```
void SetServer(core_Database *callersDb);
```

This method sets the server to be asked questions about the user. If this is not called, all servers are asked, total charges are totaled, and other values are minimized.

---

#### `GetUserName`

```
const char *GetUserName();
```

This function gets the user's name.

**GetUserId****const char \*GetUserId();**

This function gets the user's ID.

---

**GetTotalCharge****DWORD GetTotalCharge();**

This function gets the total charge. This is the amount charged cumulatively for the user.

---

**GetMaxCharge****DWORD GetMaxCharge();**

This function gets the maximum charge. This is the total amount that can be charged for the user; that is, the charge up to the account balance.

---

**GetNumberCurrentlyLogged****WORD GetNumberCurrentlyLogged();**

This function gets the number of users currently using this username account.

---

**GetTotalLogins****WORD GetTotalLogins();**

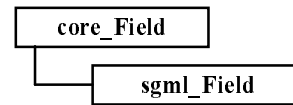
This function returns the total number of times this account has been used.



## Class `sgml_Field`

**See Also:** `sgmlfld.hpp`

The class `sgml_Field` provides standard generalized markup language (SGML) format for searching full content databases (the full content of an article is retrieved).



## Public Member Functions

The following are public member functions of class `sgml_Field`:

---

### `sgml_Field`

```
sgml_Field(core_Database *theDb = 0);
```

```
sgml_Field(const core_Field &from);
```

The class constructors.

---

### `~sgml_Field`

```
virtual ~sgml_Field();
```

The class destructor.

---

### `SetFieldName`

```
void SetFieldName(const prString &theFieldName);
```

This function sets the name of the field.

---

### `GetContentType`

```
prString GetContentType();
```

This function gets the content type.

---

### `GetContentField`

```
prString GetContentField();
```

This function gets the contents of a field.

## Class `www_Admin`

**See Also:** File `wwwadmin.hpp`

The class `www_Admin` provides macros that set and get administrative information from an ERL server. This information is used with the usage based pricing (UBP) WebSPIRS client.

`www_HTML_Helper`

`www_Admin`

## Public Member Functions

Following are the public member functions of class `www_Admin`:

---

### `www_Admin`

`www_Admin(www_Request *callersRequest);`

The class constructor.

---

### `~www_Admin`

`virtual ~www_Admin();`

The class destructor.

---

### `GetCostPerAbstract`

`static const prString &GetCostPerAbstract (const char *callersMacro, www_HTML_Helper *callersThis);`

This function provides the cost of a record using the abstract (AB) field.

## Class **www\_Alert**

**See Also:** File `wwwalert.hpp`

The class `www_Alert` provides macros for the selective dissemination of information (SDI) alerting service. It provides information about new records.

### Public Member Functions

Following are the public member functions of class `www_Alert`:

---

#### **www\_Alert**

```
www_Alert(www_Request *callersRequest);
```

The class constructor.

---

#### **~www\_Alert**

```
virtual ~www_Alert();
```

The class destructor.

---

#### **AddAlert**

```
static const prString &AddAlert (const char *callersMacro, www_HTML_Helper *callersThis);
```

This function causes the search to be run as an alert.

---

#### **EvaluateAlert**

```
static const prString &EvaluateAlert (const char *callersMacro, www_HTML_Helper *callersThis);
```

This function finds out if there are new records for the search.

---

#### **Reset**

```
void Reset();
```

This function cleans up the results of an alert.

## Class `www_Arguments`

**See Also:** File `wwwhelper.hpp`

The class `www_Arguments` stores a block of text and interprets the block of text. It is used with nested macros which are included as arguments to other macros by way of the `[SP_BLOCK] ...[/SP_BLOCK]` tags.




---

### `www_Arguments`

`www_Arguments();`

`www_Arguments(const www_Arguments &from);`

The class constructors.

---

### `~www_Arguments`

`virtual ~www_Arguments();`

The class destructor.

## Public Methods

Following are the public methods of class `www_Arguments`:

---

### `operator=`

`const www_Arguments &operator=(const www_Arguments &from);`

The assignment operator.

---

### `SetRequest`

`void SetRequest(www_Request *theRequest);`

Makes the request available to arguments for processing.

---

### `GetRequest`

`www_Request *GetRequest() const`

Gets the request set by `SetRequest`.

---

### `AddExpandTag`

`void AddExpandTag(const prString expandTag);`

This method copies your tag into a buffer (if a buffer is supplied) and returns a pointer to the item. It can give you a number or string and supply the default value.

**GetItem**

```
const void *GetItem(const char *Tag, void *Buffer = 0, uint Buflen = 0) const;
```

```
ulong GetItem(const char *Tag, ulong DefaultValue) const;
```

```
const char *GetItem(const char *Tag, char *DefaultValue) const;
```

Copies your *tag* into a *buffer* if a buffer is supplied. It returns a pointer to the item, a number, or a string. If a `[sp_block]...[/sp_block]` is being processed, the contents of the “block” are interpreted as if it were a template file.

## Class `www_Database`

**See Also:** Files `wwwdb.hpp`    `wwwdb.cpp`

The `www_Database` class provides methods for expanding database lists, adding title screens, and opening the database(s) needed by the request. This class is for SP internal use only.



### Public Member Functions

The following are public member functions of class `www_Database`:

---

#### `www_Database`

```
www_Database(www_Request *callersRequest);
```

The class constructor.

---

#### `~www_Database`

```
virtual ~www_Database();
```

The class destructor.

---

#### `GetDatabase`

```
virtual core_Database *GetDatabase()
```

Gets the specified database.

---

#### `Reset`

```
virtual void Reset();
```

Freezes any memory being used by the class.

### HTML Expansion Methods

The following methods expand database elements to HTML in the template.

---

#### `GetDatabaseName`

```
const prString &GetDatabaseName(const char *callersMacro = 0);
```

This method uses the `sp.dbname` macro to expand the database name to HTML.

## DatabaseForEach

```
const prString &DatabaseForEach(const char *callersMacro = 0);
```

This method uses the `sp.avail.dbs.foreach` macro to expand the database list to HTML.

---

## GetDatabaseDescriptions

```
const prString &GetDatabaseDescriptions(const char *callersMacro = 0);
```

This method uses the `sp.database.descriptions` macro to expand the database(s) descriptions to HTML.

---

## GetDatabaseTitleScreens

```
const prString &GetDatabaseTitleScreens(const char *callersMacro = 0);
```

Expands the title screen(s) of the database(s) to HTML.

---

## GetListForEach

```
const prString &GetListForEach(const char *callersMacro = 0);
```

This method uses the `sp.opened.dbs.foreach` macro to expand the database checklist to HTML.

---

## GetDatesCovered

```
const prString &GetDatesCovered(const char *callersMacro = 0);
```

Expands the dates covered by the database to HTML.

---

## GetDatabaseTag

```
const prString &GetDatabaseTag(const char *callersMacro = 0);
```

This method inserts the contents of a database information tag into the form.

---

## MakeWinspirsLink

```
const prString &MakeWinspirsLink(const char *callersMacro = 0);
```

Expands the WinSPIRS link to HTML.

## **BuildAllFieldSets**

**static void BuildAllFieldSets(core\_Database \*theDb);**

This is an internal routine which builds the fieldsets that the WebSPIRS interface expects. These are:

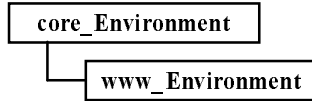
- \*R – The list of “reference fields” for full text databases.
- \*H – The include fields with the hits “field”
- \*A – The all fields fieldset
- \*F – The free text fields fieldset
- \*SEPARATE – The fields with a separate index. Used for the index display



## Class `www_Environment`

**See Also:** `wwwenv.hpp`

The `www_Environment` class performs chores necessary to start the Content Operative Retrieval Engine (CORE) and creates and provides global access to the list of available databases.



### Public Member Functions

Following are the public member functions of class `www_Environment`:

---

#### `www_Environment`

```
www_Environment(const char *ConfigFile = 0, const char *errMsgFile = 0, const char *debugFile = 0,
uint IdleRate = HIGHVAL(uint));
```

The class constructor.

---

#### `~www_Environment`

```
virtual ~www_Environment();
```

The class destructor.

---

#### **Start**

```
virtual void Start();
```

After constructing the class, the application must call **Start** to initialize memory.

---

#### **Idle**

```
virtual boolean www_Environment::Idle(boolean Force);
```

This function was for the WINDOWS environment, but is no longer used.

---

#### **Assert**

```
virtual void Assert(const char *filename, uint linenumber, const char *exp = 0);
```

Use this function if your environment is defined as WIN32. If a program encounters a fatal condition, call **Assert**. The program will display a message and then die.

**GetCurrentDatabaseList**

```
core_CoreDBList *GetCurrentDatabaseList();
```

This function returns the current database list.

---

**GetDatabaseList**

```
core_CoreDBList *GetDatabaseList();
```

This function returns the database list after checking that the `erlclnt.cfg` rebuild the list if it has.

---

**RemoveDatabaseList**

```
void RemoveDatabaseList();
```

This function removes the database list so you can build a new one.

---

**SetRequest**

```
void SetRequest(www_Request *thisRequest)
```

This function sets the request in *thisRequest*.

---

**GetPtr**

```
static www_Environment *GetPtr();
```

This function gets a pointer to the `www_Environment` class and can get a database list from it.

---

**GetERLPath**

```
prString GetERLPath();
```

This function gets the ERL path.

## Class `www_ERLConnection`

**See Also:** `wwwerl.hpp`

The class `www_ERLConnection` provides the login functions for the ERL connection.

`core_ERLConnection`

`www_ERLConnection`

## Public Member Functions

Following are the public member functions of class `www_ERLConnection`:

---

### `www_ERLConnection`

**`www_ERLConnection(core_Config &config, const char *erlCfgPath, const char *logFile = 0);`**

The class constructor. This will create a log file if *logFile* is not equal to zero. If the first character of the filename is '+', append mode opens.

---

### `~www_ERLConnection`

**`virtual ~www_ERLConnection();`**

The class destructor.

---

### Refresh

**`virtual boolean Refresh();`**

This function checks to see if the database list has changed and rebuilds it and returns `cTRUE` if it has.

---

### LoginFailed

**`boolean LoginFailed() const;`**

This function provides information when a login fails; for example, when an incorrect username and/or password is entered.

---

### FatalErrorOccurred

**`boolean FatalErrorOccurred() const;`**

This function provides information when a fatal error occurs; that is, when an unrecoverable error occurs.

---

### SetClientAddress

**`void SetClientAddress(const prString &callersClientAddress);`**

Sets the information provider's address--the *callersClientAddress*.

**GetClientAddress****const prString &GetClientAddress()**

Returns the information provider's address set by **SetClientAddress**.

---

**PasswordExpired****boolean PasswordExpired() const**

WebSPIRS calls this function to find out if the tells the password has expired.

---

**GetExpiredServer****const prString &GetExpiredServer() const**

This function returns information about the expired server.

---

**SetRequest****void SetRequest(www\_Request \*theRequest)**

This function sets the request in *theRequest*.

---

**SetNewPassword****const void SetNewPassword(const prString &newPassword)**

This function sets the password in *newPassword*.

---

**SetConfirmPassword****const void SetConfirmPassword(const prString &newPassword)**

This function confirms the password in *newPassword*.

---

**GetErrorMessage****const prString &GetErrorMessage() const;**

This function returns an error message.

---

**SetErrorMessage****www\_ERLConnection &SetErrorMessage(const prString &callersErrorMessage);**

This function sets the error message in *callersErrorMessage*.

---

## GetSelf

```
static www_ERLConnection *GetSelf()
```

This function returns the caller's ERL connection.

## Protected Member Functions

Following are the protected member functions of class `www_ERLConnection`:

---

### ShowMessage

```
virtual void ShowMessage(const char *servername, const char *msg);
```

This function displays a message from the server. This need not be an error message.

---

### Login

```
virtual uint Login (const char *servername, const char *serverid);
```

The server calls to get the username and password. This function returns 0 if they are correct; otherwise, it returns `cLOGIN_CANCEL`. The username and password should be put in the appropriate `prStrings` of `SetNewPassword` and `SetConfirmPassword`.

---

### BadUser

```
virtual uint BadUser(const char *servername, const char *serverid);
```

This function displays a login error message which comes back from the server. It may call `Login` to retry if desired; otherwise, it returns `cLOGIN_CANCEL`.

---

### PasswordExpired

```
virtual uint PasswordExpired(const char *servername, const char *serverid);
```

This function displays a login error message which comes back from the server. It may call `Login` to retry if desired; otherwise, it returns `cLOGIN_CANCEL`.

---

### MaxUsers

```
virtual uint MaxUsers(const char *servername, const char *serverid);
```

This function informs the user that ERL has reached the maximum number of users. There is no point in redoing the login; they can try again later.

## DXPError

**virtual void DXPError(const char \*servername, const char \*dbid, const char \*dbname, ulong reqID, boolean error, uint dxs\_error, const char \*usermsg, const char \*devmsg);**

This function returns various error messages.

---

## ProtocolError

**virtual void ProtocolError(const char \*servername, uint error\_code, const char \*msg);**

This function returns an error message which has been formatted using the CORE's **LastErrorMsg()** interface. If not properly initialized (see the **mmEnvironment** class in the *SilverPlatter CORE Wrapper Reference Manual*), it will be blank.

---

## ConnectionDied

**virtual boolean ConnectionDied (const char \*servername, const char \*dbid, const char \*dbname, boolean can\_retry);**

If *can\_retry* is cTRUE, the network layer should try to reestablish the connection.

## Class `www_Field`

**See Also:** `wwwfield.hpp`

The class `www_Field` expands field table type requests into lists of fields.

`www_HTML_Helper`

`www_Field`

## Public Member Functions

Following are the public member functions of class `www_Field`:

---

### `www_Field`

`www_Field(www_Request *callersRequest);`

The class constructor.

---

### `~www_Field`

`virtual ~www_Field();`

The class destructor.

---

### `GetFieldList`

`const prString &GetFieldList(const char *callersMacro = 0);`

This function produces a list of fields that are hardwired in the option list format.

---

### `GetFieldListForEach`

`const prString &GetFieldListForEach(const char *callersMacro = 0);`

This function produces a list of fields. Using a macro, the items in the list can be turned into checkbox or radio button items. This method is preferred over `GetFieldList`.

## Class `www_FSI`

**See Also:** File `wwwfsi.hpp`

The class `www_FSI` provides field specific index (FSI) type requests. This class is for SP internal use only.



### Public Member Functions

The following are public member functions of class `www_FSI`:

---

#### `www_FSI`

```
www_FSI(www_Request *      );
```

The class constructor.

---

#### `~www_FSI`

```
virtual ~www_FSI();
```

The class destructor.

---

#### `Reset`

```
void Reset();
```

This method deletes the `core_DictionaryWord` used by the `FSIList` code.

---

#### `GetFSIList`

```
static const prString &GetFSIList(const char *callersMacro, www_HTML_Helper *callersThis);
```

This method produces a list of words based on the criteria. It uses the `sp.fsi.list` macro.

---

#### `FSIToSearch`

```
static const prString &FSIToSearch(const char *callersMacro, www_HTML_Helper *callersThis);
```

This method produces a list of words based on the criteria. It uses the `sp.fsi.tosearch` macro, which displays the dictionary words for a field-specific index.

---

#### `DoFSIToc`

```
static const prString &DoFSIToc(const char *callersMacro, www_HTML_Helper *callersThis);
```

This method uses the `sp.fsi.toc` macro and produces a list of words based on the criteria.



## CopyToVariable

```
static const prString &CopyToVariable(const char *callersMacro,           *callersThis);
```

This method uses the `sp.fsi.copytovariable` macro and copies the contents of a field-specific index (FSI) to a specified variable. This allows FSIs to be turned into Select lists.

---

## PreprocessRequest

```
virtual void PreprocessRequest(core_Config & templateArguments);
```

This method handles `next` and `prev` index scrolling.

## Class `www_Guide`

**See Also:** File `wwwguide.hpp`

The class `www_Guide` provides the to hold any macros needed to access SilverPlatter guides from a template. This class is for SP internal use only.

`www_HTML_Helper`

`www_Guide`

## Public Member Functions

The following are public member functions of class `www_Guide`:

---

### `www_Guide`

`www_Guide(www_Request *callersRequest);`

The class constructor.

---

### `~www_Guide`

`virtual ~www_Guide();`

The class destructor.

---

### `MakeGuideToc`

`const prString &MakeGuideToc(const char *callersMacro);`

This method uses the `sp_guide.toc` macro which displays the guide table of contents.

---

### `MakeGuideTopic`

`const prString &MakeGuideTopic(const char *callersMacro);`

This method uses the `sp_guide.topic` macro which inserts the text for the guide topic.

---

### `GetGuideDBName`

`const prString &GetGuideDBName(const char *callersMacro);`

This method uses the `sp_guide.dbname` macro which is replaced with the database name of the guides that are being expanded.

## Class `www_HTML_Helper`

**See Also:** Files `wwwhelpr.hpp` and `wwwhelpr.cpp`

The base class `www_HTML_Helper` provides base support for interpreting macros and expanding them in HTML. It provides routines for building the contents of an option list. It also provides lookup for searching the macros owned by the class and calling the appropriate expansion macros on a match. This class uses the `www_Request` class for access to the values of tags and global items like the database.

### Public Member Functions

The following are public member functions of class `www_HTML_Helper`:

---

#### `www_HTML_Helper`

```
www_HTML_Helper(www_Request *callersRequest);
```

The class constructor.

---

#### `~www_HTML_Helper`

```
virtual ~www_HTML_Helper();
```

The class destructor.

---

#### `Init`

```
void Init(wwwMacroDef *callersMacros);
```

Provides a table of macro handlers for the class to use in expanding macros coded in the template. The last entry should be 0,0.

---

#### `GetName`

```
const prString &GetName() const;
```

Gets the name of the helper.

## SetName

`www_HTML_Helper &SetName(const prString & callersName);`

Set by the static helper creator template class.

## Macro Expansion Methods

The following are macro expansion methods of the class `www_HTML_Helper`:

---

### Lookup

`virtual const prString &Lookup(const char *key);`

Expands *key* into the requisite HTML; each helper need only implement those keys it knows about.

---

### GetKeyFoundFlag

`boolean GetKeyFoundFlag() const;`

Lets the caller know whether the key was processed, since `prNULLSTRING` is a valid return from **Lookup**.

---

### SetKeyFoundFlag

`www_HTML_Helper &SetKeyFoundFlag(boolean callersKeyFoundFlag);`

Internal method for determining that a macro expansion has taken place.

---

## Utility Methods

The following are utility methods of class `www_HTML_Helper`:

---

### PreprocessRequest

`virtual void PreprocessRequest(core_Config & callersArguments);`

Allows helpers to interpret things like `sp.record.action` before the form is interpreted.

---

### GetRecords

`virtual core_RecordArray *GetRecords(const prString & callersSource);`

Allows the helpers that know how to make record arrays (`www_Search` and `www_FSI`) to do so.

---

## GetDatabase

**virtual core\_Database \*GetDatabase();**

Allows helpers that know how to open databases (**www\_Database**) to do so.

---

## Reset

**virtual void Reset();**

Cleans things up for a subsequent request.

---

## GetHiddenVariables

**const prString &GetHidden Variables(const prStringArray &VariablesFormSets);**

Gets `<input type="hidden" name="variable">` strings required to preserve this one's context.

---

## LoginRequired

**boolean LoginRequired(const char \*callersTag);**

Finds out if any of our macros require a true ERL login.

---

## GetMacroCount

**uint GetMacroCount() const**

Allows external classes to get the helper's macros for purposes of producing a nicely HTML-formatted list of macros.

---

## GetMacro

**const www\_Macro \*GetMacro(uint which) const**

Provides a set of macros this helper is prepared to support.

---

## GetRequest

**www\_Request \*GetRequest()**

Provides access to the request.

---

## GetExpansionString

**prString &GetExpansionString()**

Returns the work string to use in the expansion.

---

## InterpretArguments

**gift cgi\_Config \*InterpretArguments(const char \*callersMacroArguments);**

Interprets the keys from a macro.

---

## GetDescription

**const prString &GetDescription() const;**

Gets a description of the macros contained by the helper. This method is no longer used. It was included when the intention was to have the macros be self-documenting.

---

## SetDescription

**www\_HTML\_Helper &SetDescription(const prString &callersDescription);**

This method sets a description of the macros. It is also not used.

## Utility Methods for Generating HTML

The following are utility methods of class `www_HTML_Helper` for generating HTML:

---

### MakeOptionHTML

**prString MakeOptionHTML(const char \*callersValueArg , const char \*callersText, const prString &callersSelections);**

This method makes a `<OPTION value="callersValueArg">callersText</OPTION>` entry. It will insert the `SELECTED` attribute if *callersValueArg* is a comma-delimited string found in *callersSelections*.

**prString MakeOptionHTML(const char \*callersValueArg , const char \*callersText, boolean callersSelection = cFALSE);**

This methods makes a `<OPTION value="callersValueArg">callersText</OPTION>` entry. It will insert the `SELECTED` attribute if *callersSelection* is `cTRUE`.

---

### MakeCheckboxHTML

**prString MakeCheckboxHTML(const char \*callersVariableName , const char \*callersValue, const char \*callersText, const prString &callersSelections);**

This method makes a checkbox: `<INPUT TYPE="CHECKBOX" NAME="callersVariableName" VALUE="callersValue">callersText</INPUT>`. The checkbox is `CHECKED` if the value matches one of the comma-delimited values in *callersSelections*.

**prString MakeCheckboxHTML(const char \*callersVariableName , const char \*callersValue, const char \*callersText, boolean callersChecked = cFALSE);**

This method makes a checkbox: `<INPUT TYPE="CHECKBOX" NAME="callersVariableName" VALUE="callersValue">callersText</INPUT>`. The checkbox is `CHECKED` if the value of *callersChecked* is `cTRUE`.

## MakeHyperLink

**prString MakeHyperLink(const prString &callersPrompt , const prString &callersHREF);**

This method makes the <A HREF="">prompt</A> markup.

---

## MakeUrl

**prString MakeUrl(const prString &callersDestinationForm , const prString &callersDbId = prNullString, const prString &callersExtraArguments = prNullString);**

This method creates a URL to WebSPIRS aimed at the specific form. This will include certain defined bits of context:

sp.usernumber.p &sp.dbid.p

Anything else can be added in extra arguments as tag=value&tag=value&... The *callersDbId* argument is left prNullString. The sp.dbid.p= will default to the currently open database.

---

## LoadGatewayFile

**prString LoadGatewayFile(const prString &callersFilename, prString &callersReturnFileName , prString &callersReturnTemplateArguments);**

This method loads a file determined by *callersFilename*. This may be an entry in the request associative array, in which case it is expanded. It puts *callersFilename* in *callersReturnFileName* and any arguments in *callersReturnTemplateArguments*.

---

## Table of Helpers Methods

The following set of methods are needed to construct the table of all helpers. Each implemented **www\_HTML\_Helper** class should implement a hidden static class using the C++ template class **www\_HelperConstructor** <class>which calls **Initialize** and **AddHelperConstructor** in its constructor and **Terminate** in its destructor. The **www\_Request** class calls **ConstructHelpers** once to build the table of the HTML helpers needed to expand the macros.

---

## Initialize

**static void Initialize();**

This method prepares the table.

---

## Terminate

**static void Terminate();**

This method destroys the table. The table is actually only freed when all the helpers have been terminated.

---

## GetHelperCount

```
static uint GetHelperCount();
```

This method returns a count of the helpers.

---

## AddHelperConstructor

```
static void AddHelperConstructor(www_HTML_Helper* (*HelperConstructor)(www_Request  
*callersRequest));
```

This method adds this helper's constructor to the table.

---

## ConstructHelpers

```
static gift www_HTML_Helper **ConstructHelpers(www_Request *callersRequest);
```

This request is called once to build a table of helpers.

---

## Protected Methods

The following are protected methods of class `www_HTML_Helper`:

---

### SetCount

```
void SetCount(uint callersMacroCount);
```

Sets a macro count.

---

### SetMacro

```
void SetMacro(uint which, www_Macro *callersMacro);
```

Constructs a new macro.



## Class `www_Macro`

**See Also:** File

The `www_Macro` class connects the outside world (the `www_Template` class) with the specific macro expansion routine (if any) for the macro. This linkage happens through a static method in the `www_HTML_Helper` derived class. See class `www_Database` for examples of how this is done.

The purpose of the `www_Macro` class is to define the aspects of individual macros, which are:

Its name.

A description for use in automatically generating documentation.

Whether the macro is part of the “state”, that is, whether it is a variable and should have an `<input type="hidden" generated for it.`

Through the subsequent template class, whether and how the macro may be expanded to HTML.

## Public Member Functions

The following are public member functions of class `www_Macro`:

---

### `www_Macro`

`www_Macro();`

A class constructor.

`www_Macro(const www_Macro &from);`

A copy constructor.

---

### `~www_Macro`

`virtual ~www_Macro();`

The class destructor.

---

### `operator=`

`const www_Macro &operator = (const www_Macro &from);`

The assignment operator.

---

### `GetName`

`const prString &GetName() const;`

Returns the name of the macro; matched against the contents of `[SP_MACRO] . . . [/SP_MACRO]`.

---

### `SetName`

`www_Macro &SetName(const prString &callersDescription);`

Sets the name of the macro. For example, it could set the name of the macro by using `"sp.record.fieldtext"` *callersName* parameter.

## SetDescription

```
www_Macro &SetDescription(const prString &callersDescription);
```

Sets the macro description.

---

## GetDescription

```
const prString &GetDescription() const;
```

Gets the macro description.

---

## IsContextVariable

```
boolean IsContextVariable() const;
```

The variable must be preserved if it has been assigned a value and there are no HTML input/ selects for it.

---

## AlwaysGenerateHidden

```
boolean AlwaysGenerateHidden() const;
```

This returns cTRUE if hidden inputs should be generated for those portions of the “.pp” type variable’s value that were not supplied by <INPUT> statements in the template.

---

## LoginRequired

```
boolean LoginRequired() const;
```

Is a real login required to use this macro?

---

## SetLoginRequired

```
www_Macro &SetLoginRequired(boolean callersLoginRequired);
```

Returns cTRUE if the user must be logged into ERL before the macro can run.

---

## ExpansionMethod

```
const prString &(*ExpansionMethod)(const char *callersMacro, www_HTML_Helper *callersHelper);
```

This method points to a **www\_HTML\_Helper** method which does the work of expanding the macro:

---

## GetExpansionMethod

```
ExpansionMethod GetExpansionMethod() const
```

Gets the expansion method.

---

## SetExpansionMethod

```
www_Macro &SetExpansionMethod(ExpansionMethod callersExpansionMethod);
```

Sets the expansion method.

---

## Class `www_MacroCaller`

See Also: Files `wwwmacro.h` and `wwwmcall.cpp`

The `www_MacroCaller` class connects the C++ classes to macros implemented in C. These macros are defined in `wwwmacro.h`.



### Public Member Functions

The following are public member functions of class `www_MacroCaller`:

---

#### `www_MacroCaller`

`www_MacroCaller(www_Request *callersRequest);`

The class constructor.

---

#### `~www_MacroCaller`

`virtual ~www_MacroCaller();`

The class destructor.

---

#### Lookup

`virtual const prString &Lookup(const char *key);`

Expands *key* into the requisite HTML; each helper need only implement those keys it knows about.

## Class `www_Record`

**See Also:** Files `wwwrecrd.hpp` and `wwwrecrd.cpp`

The `www_Record` class provides methods for inserting record text. Its purpose is to fulfill record extraction type macros using a `core_RecordArray` supplied by one of the other helpers.

`www_HTML_Helper`

`www_Record`

### Public Member Functions

The following are public member functions of class `www_Record`:

---

#### `www_Record`

```
www_Record(www_Request *callersRequest);
```

The class constructor.

---

#### `~www_Record`

```
virtual ~www_Record();
```

The class destructor.

---

#### `Reset`

```
void Reset();
```

Freezes any memory being used by the class.

### Record Extraction Methods

The following are the internal record extraction methods of class `www_Record`:

---

#### `GetRecordCounts`

```
const prString &GetRecordCounts(const char *callersMacro = 0);
```

Returns the count of the records.

---

#### `GetRecordText`

```
const prString &GetRecordText(const char *callersMacro = 0);
```

Returns the text of the record.

---

#### `GetRecordsExtracted`

```
const prString &GetRecordsExtracted(const char *callersMacro = 0);
```

Gets the records that were extracted.

## GetRecordNumber

```
const prString &GetRecordNumber(const char *callersMacro = 0);
```

Returns the record number.

---

## GetHowMany

```
const prString &GetHowMany(const char *callersMacro = 0);
```

Returns the number of displayed records.

---

## GetTableOfContents

```
const prString &GetTableOfContents(const char *callersMacro = 0);
```

This method uses the `sp.record.toc` macro to get the table of contents.

---

## GetRecordFieldText

```
const prString &GetRecordFieldText(const char *callersMacro = 0);
```

This method uses the `sp.record.fieldtext` macro which displays text for a range of fields.

---

## GetRecords

```
core_RecordArray *GetRecords(const prString &callersSource = 0);
```

This method is used for following a hotlink.

---

## SortRecords

```
core_RecordArray *SortRecords(core_RecordArray *callersRecords);
```

If the value of the `sp.record.sortrecords.p` macro is “Yes,” this method sorts the current set of records.

---

## PreprocessRequest

```
virtual void PreprocessRequest(core_Config & templateArguments);
```

This method handles the `sp.record.action` macro and is called before form expansion takes place.

---

## RecordInitialize

```
static const prString &RecordInitialize(const char *callersMacro, www_HTML_Helper *callersThis);
```

Calculates all the parameter values such as the number of records to show, the first record, and so forth.

## GetCurrentRecordValue

```
static const prString &GetCurrentRecordValue(const char *callersMacro, www_HTML_Helper
*callersThis);
```

This method is connected to the `sp.currentrecord.field` macro. It gets the text for the current record, as defined by the variable, `sp.currentrecord`. The extraction is defined by the argument `FIELDS`. The text is converted from DXP to HTML.

---

## DoURL

```
static const prString &DoURL(const char *callersMacro, www_HTML_Helper *callersThis);
```

This method is connected to the macro variable, `sp.url.p`. It handles the expansion of the contents of that variable. The contents are defined in the CORE wrapper header file, `crspurl.hpp`. This routine is intended to serve when WebSPIRS is being used without a specified template, for example:

```
...\webspirs.bat?sp.usernumber=1,cmd="yes",sp.url.p=I(RING)F(*BOS=#1)
```

would deliver a TIFF file from a fictitious RING document database. The expansion will take one of the following paths:

- If the URL specifies an external file; for example, a GIF file, the contents of the file are read and delivered with the appropriate content type as defined by the database information.
- If the URL specifies a field of DXP text or a DXP binary hotlink, then the text/hotlink is extracted and the content type is defined by the database information.

---

## GetCurrentRecordText

```
static const prString &GetCurrentRecordText(const char *callersMacro, www_HTML_Helper
*callersThis);
```

This method uses the `sp.currentrecord.text` macro which converts the text in a specified field to HTML.

---

## GetCurrentPlainText

```
static const prString &GetCurrentPlainText(const char *callersMacro, www_HTML_Helper
*callersThis);
```

This method formats the current record based on WebSPIRS default HTML markup, which is a `<DL><DT><DD>...</DL>` type list.

---

## GetCurrentDBID

```
static const prString &GetCurrentDBID(const char *callersMacro, www_HTML_Helper *callersThis);
```

This method provides the database ID of the current record (`sp.currentrecord`).

## GetCurrentDBName

```
static const prString &GetCurrentDBName(const char *callersMacro, www_HTML_Helper  
*callersThis);
```

This method provides the database name of the current record (`sp.currentrecord`).

---

## GetCurrentRecordURL

```
static const prString &GetCurrentRecordURL(const char *callersMacro, www_HTML_Helper  
*callersThis);
```

This method builds a URL to the current record. This is used in the usage-based pricing (UBP) WebSPIRS.

---

## GetAbstractCostPerRecord

```
static const prString &GetAbstractCostPerRecord(const char *callersMacro, www_HTML_Helper  
*callersThis);
```

This method gets the cost per record for the database from which the current record was retrieved. Used by UBP WebSPIRS.

---

## MakeSPUrl

```
static const prString &MakeSPUrl(const char *callersMacro, www_HTML_Helper *callersThis);
```

This method generates the `sp.url.p` portion of a WebSPIRS hotlink.

---

## BetweenTOC

```
static const prString &BetweenTOC(const char *callersMacro, www_HTML_Helper *callersThis);
```

This method generates the between record (“inter record”) table of contents.

---

## GetRecordField

```
static const prString &GetRecordField(const char *callersMacro, www_HTML_Helper *callersThis);
```

This method gets a field's text and is used to display the text for full content.

---

## GetLastURL

```
const prString &GetLastURL();
```

This method returns the last `sp.url.p` expanded by the **DoURL** method.

---

## Class `www_Request`

**See Also:** Files `request.hpp` and `request.cpp`

The `www_Request` class handles the details of interpreting a HTTP request. The various other classes all get a pointer to it and use it to get the values of the various macros. This is done using the `core_Config` API (see the *SilverPlatter CORE Wrapper Reference Manual* or the `config.hpp` header file).



This class offers `GetDatabase()` to specific methods, which returns a pointer to the `core_Database` object (see the *SilverPlatter CORE Wrapper Reference Manual* or the `crvdb.hpp` header file). This pointer is invaluable in database-specific processing.

The `www_Request` class makes sure the user is properly logged into the ERL server(s). If not, it generates a special form, `login.htm`, using the `www_Template` class and sends this back as the filled in form. This will prompt the user to supply a valid ERL username and password.

The `www_Request` class uses the following macros:

- `sp.username` and `sp.password`, which are supplied to the ERL connection class, `core_ERLConnection`.
- `sp.nextform`, which determines the template form to be loaded. If this macro is absent, the request goes into command mode (for use by Perl scripts); otherwise, the request goes to the `www_Template` class as described above.

## Public Member Functions

The following are the public member functions of class `www_Request`:

---

### `www_Request`

`www_Request();`

A class constructor.

`www_Request(const www_Request &from);`

A copy constructor.

---

### `~www_Request`

`virtual ~www_Request()`

The class destructor.

---

### `operator=`

`const www_Request &operator= (const www_Request &from);`

The assignment operator.



## SetRequest

```
void SetRequest(const char *buffer);
```

Contents of buffer will get unpacked, so it gets modified.

---

## Reset

```
void Reset();
```

Cleans things up after a request has been processed.

---

## IsTemplateDriven

```
boolean IsTemplateDriven();
```

If no template is to be expected, you are being called (via Perl for example).

---

## GenerateForm

```
const prString &GenerateForm();
```

Returns the filled out HTML form.

---

## DoCommands

```
const prString &DoCommands();
```

Returns text generated by the tags, that is, no forms.

---

## GetHiddenVariables

```
prString &GetHiddenVariables(prStringArray & VariablesFormSets);
```

Gets `<input type="hidden" name="variable">` strings required to preserve this one's context.

---

## RunMacro

```
const prString &RunMacro(const prString &theMacro);
```

This method is used to expand a macro outside of the context of a template. This happens in the following circumstances:

- When WebSPIRS is processing a URL that does not specify a template (delivering a non-HTML hotlink primarily).
- When WebSPIRS encounters a SP macro in database text. This happens when brave souls put SP macros in the library holdings message.
- When WebSPIRS is expanding the ``{`` syntax used to build search terms.

**GetHelperCount****uint GetHelperCount();**

Returns the number of helpers in the request.

---

**GetHelper****www\_HTML\_Helper \*GetHelper(uint which);**

Gets a pointer to a helper.

---

**DbIRefresh****boolean DbIRefresh();**

This method checks to see if the database list needs refreshing prior to further processing, particularly opening a database. This list will need refreshing if the user name, password, or IP address has changed.

---

**GetDatabase****core\_Database \*GetDatabase();**

Gets a pointer to the database being used by the request.

---

**GetRecords****core\_Search \*GetRecords(const prString & callersHelperName =prNullString);**

Gets a pointer to the records to be displayed by the request.

---

**ConvertConcatSymbol****const prString &ConvertConcatSymbol(const char \*callersValue, char callersConcatSymbol =',');**

This method is no longer used.

---

**GetErrorMessage****const prString &GetErrorMessage()**

Gets the error message that describes any problems encountered in processing the macro.

## Class `www_Search`

**See Also:** `wwwsrch.hpp` and `wwwsrch.cpp`

The `www_Search` provides methods which do the following:

- Hang onto the search for the length of the request.
- Interpret the automatic subject lookup (ASL) macro.
- Get the parsed search text or reports an error.
- Do the thesaurus tasks.



This class is for SP internal use only.

## Public Member Functions

The following are public member functions of class `www_Search`:

---

### `www_Search`

```
www_Search(www_Request *callersRequest);
```

The class constructor.

---

### `~www_Search`

```
virtual ~www_Search();
```

The class destructor.

---

### Reset

```
void Reset();
```

Freezes any memory being used by the class.

---

### GetRecords

```
core_RecordArray *GetRecords(const prString & callersSource);
```

This method builds a search based on the contents of the `sp.search.value.p` macro and returns a pointer to the records matched by the search.

---

### GetASLList

```
static const prString &GetASLList(const char *callersMacro, www_HTML_Helper *callersThis);
```

This method handles the `sp.asl.list` macro.

## GetSearchText

```
static const prString &GetSearchText(const char *callersMacro, www_HTML_Helper *callersThis);
```

This method handles the `sp.record.search` macro.

---

## GetTermDefinition

```
static const prString &GetTermDefinition(const char *callersMacro, www_HTML_Helper *callersThis);
```

This method handles the `sp.term.definition` macro.

---

## PrepareTerm

```
static const prString &PrepareTerm(const char *callersMacro, www_HTML_Helper *callersThis);
```

This method handles the `sp.term.prepare_detail` macro.

---

## DoPermutedList

```
static const prString &DoPermutedList(const char *callersMacro, www_HTML_Helper *callersThis);
```

This method handles the `sp.permuted_list` macro.

---

## BuildSearchHistory

```
static const prString &BuildSearchHistory(const char *callersMacro, www_HTML_Helper *callersThis);
```

This method handles the `sp.searchhistory.build` macro. It takes the current search and adds it to the search history.

## Class `www_Server`

**See Also:** File `wwwservr.hpp`

The class `www_Server` gets the requests from the common gateway interface (CGI) server, parses the requests, and acts on them. Input requests are handled by the hypertext transport protocol (HTTP) and the output is a hypertext markup language (HTML) form.

### Public Member Functions

Following are the public member functions of class `www_Server`:

---

#### `www_Server`

`www_Server();`

The class constructor.

---

#### `~www_Server`

`virtual ~www_Server();`

The class destructor.

---

#### `Go`

`virtual void Go();`

The function consists of the following loop:

```
while (GetRequest)
{
    GenerateForm(); // Every request is guaranteed to produce a form of some kind.
    WriteForm();
}
```

Default requests come from the `.cfg` file tag `WWW.REQUEST_FILE`, default = `wwwservr.req`. This file should have one line in `tag=, tag=...` format. **GetRequest** reads the file whenever the date changes. You can terminate the business by deleting the file. Input requests will be echoed to `sysout` and a debug file if the `.cfg` file tag `WWW.ECHO_REQUESTS` is set. Similarly, output forms will be echoed if `WWW.ECHO_FORMS` is set.

---

#### `SetEchoRequestsFlag`

`www_Server &SetEchoRequestsFlag(boolean theEchoFlag);`

This function sets the flag that echoes requests. Useful for debugging.

---

#### `GetEchoRequestsFlag`

`boolean GetEchoRequestsFlag() const;`

This function gets the request set by **SetEchoRequestsFlag**.

## SetEchoFormsFlag

```
www_Server &SetEchoFormsFlag(boolean theEchoFlag);
```

This function sets the flag the echoes forms.

---

## GetEchoFormsFlag

```
boolean GetEchoFormsFlag() const;
```

This function gets the request set by **SetEchoFormsFlag**.

---

## SetPipeName

```
www_Server &SetPipeName(const char *callersPipeName);
```

This function sets the name of the pipe.

---

## SetFileNames

```
www_Server &SetFileNames(const char *callersInputFileName, const char *callersOutputFileName);
```

This function sets the names of files.

## Class `www_Template`

**See Also:** Files `template.hpp` and `template.cpp`

The `www_Template` class handles the details of filling in a template HTML form based on information supplied by the `www_Request` class. It reads the template name supplied in its constructor, and it does any special processing specified in the various “action” macros such as `sp.record.action`.



It uses class `prFormat` to do the form fill in processing (see the *SilverPlatter CORE Wrapper Reference Manual* or the `prformat.hpp` header file).

Basically, this class copies the template until it finds a `[SP_MACRO]...[/SP_MACRO]` and then calls a **Lookup** function supplied by the template class, `www_TemplateExpander:Lookup`. This method supplies the text to be inserted in the macro's stead.

Finally, the template class determines what hidden variables are to be inserted into the finished form. It does this by building the set of variables found in `<input>`, `<select>`, and `[SP_MACRO]`s. This is the set of variables that the form itself will produce, with or without user help. It then looks at the variables contained in the request, class `www_Request`. Any of these whose name ends in a “.p” (for persistent) suffix and that are not in the first set of variables will have `<input type="hidden"...>` generated for them.

### Rules Used in `www_TemplateExpander:Lookup` to Turn Macros into HTML

- The macro is passed the various `www_HTML_Helper` lookup routines. These expand the macro if they know how. For example, the macro `sp.database.list` is expanded by the `www_Database` class in the method `GetDatabaseList()`.
- If no expansion took place, the macro's current value in the request is inserted into the form.

## Public Member Functions

The following are public member functions of class `www_Template`:

---

### `www_Template`

```
www_Template(www_Request *theRequest);
```

```
www_Template(www_Request *theRequest, const prString &fileName);
```

The class constructors.

---

### `~www_Template`

```
virtual ~www_Template();
```

The class destructor.

---

### `SetFormContents`

```
void SetFormContents(const char *callersTemplate);
```

Allows the template to expand an “in memory” template rather than load it up from a file.

## GetForm

```
const prString &GetForm();
```

Fills out the HTML form.

## Class `www_User`

See Also: `wwwuser.hpp`

```
www_HTML_Helper
```

```
www_User
```

The `www_User` class handles CGI and server specific macros such as `sp.username`, `sp.password`, and `sp.include`. See Chapter 4 for definitions of these macros.

## Public Member Functions

Following are the public member functions for class `www_User`:

---

### `www_User`

```
www_User(www_Request *callersRequest);
```

The class constructor.

---

### `~www_User`

```
virtual ~www_User();
```

The class destructor.

---

### GetMacroList

```
const prString &GetMacroList(const char *callersMacro = 0);
```

This function returns a list of all macros.

---

### IncludeTemplate

```
const prString &IncludeTemplate(const char *callersMacro = 0);
```

This function provides the `sp.include` macro which includes a file in a template.

---

### AssignValue

```
const prString &AssignValue(const char *callersMacro);
```

This function provides the `sp.assign` macro which assigns a value to a variable.



## ForEach

**const prString &ForEach(const char \*callersMacro);**

This function provides the `sp.foreach` macro which repeatedly expands the macros contained in a specified file.

---

## IfCond

**const prString &IfCond(const char \*callersMacro);**

This function provides the `sp.if` macro which creates a conditional “if-else” programming statement.

---

## GenerateURL

**const prString &GenerateURL(const char \*callersMacro);**

This function provides the `sp.generate_url` macro which generates a URL from one template to another within WebSPIRS.

---

## GetValue

**prString GetValue(const char \*callersSyntax, const char \*\*callersEndTag = 0);**

This function provides the `sp.getvalue` macro which gets the value of a variable.

## Class `www_Wild`

**See Also:** File `wwwwild.hpp`

The class `www_Wild` expands field table type requests. It was created for the Worldwide Integrated Library of Databases (WILD) Thing project. One of the goals of that project was to support various standard generalized markup language (SGML) document type definitions (DTD).



## Public Member Functions

Following are the public member functions of class `www_Wild`:

---

### `www_Wild`

```
www_Wild(www_Request *callersRequest);
```

The class constructor.

---

### `~www_Wild`

```
virtual ~www_Wild();
```

The class destructor.

---

### `BuildSGMLToCURL`

```
static const prString &BuildSGMLToCURL(const char *callersMacro, www_HTML_Helper *callersThis);
```

This method builds a link to an SGML table of contents database.

---

### `MakeLink`

```
static const prString &MakeLink(const char *callersMacro, www_HTML_Helper *callersThis);
```

This method makes a link so that the SGML viewer (Panorama) can go back and get the DTD.

---

### `GetNotation`

```
static const prString &GetNotation(const char *callersMacro, www_HTML_Helper *callersThis);
```

For some links there are specific notations, such as “this is a .GIF” and this method gets the notation.

---

### `GetCitation`

```
static const prString &GetCitation(const char *callersMacro, www_HTML_Helper *callersThis);
```

This method returns the SGML-specific citation of the record.



## Chapter 6 - Frequently Asked Questions

---

This chapter contains frequently asked questions about WebSPIRS and the answers to those questions.

---

**Q:**

What is the WebSPIRS software?

**A:**

WebSPIRS provides a usable, simple interface that also handles complex Boolean searches. Without loading any additional software, anyone with a forms-capable browser and access to the worldwide web can use WebSPIRS to search SilverPlatter databases loaded on an ERL server.

Forms created in HTML retrieve the ERL data using WebSPIRS, which was designed using a template strategy. You can use the templates as models to create your own interface forms, or you can make a copy of a template and modify it to fit your needs.

---

**Q:**

What are the minimum requirements for WebSPIRS?

**A:**

The minimum requirements for running WebSPIRS on the Linux, Solaris, and Windows NT platforms are described in Chapter 2, *Installing and Configuring WebSPIRS*.

---

**Q:**

Can WebSPIRS be run on the same server as the ERL server software?

**A:**

Yes, WebSPIRS and ERL can be run on the same system when they are both available on the same platform. Be sure that the minimum requirements are met for both WebSPIRS and ERL on the platform you choose.

---

**Q:**

Can the WebSPIRS interface be changed?

**A:**

Yes, WebSPIRS is delivered to an ERL site with a default interface provided. System administrators with knowledge of HTML can very easily customize the look of the interface. The WebSPIRS HTML interface documents are located in the `html` directory within the WebSPIRS directory. Detailed instructions can be found in Chapter 3, *Tutorial for Customizing Templates*.

**Q:**

How can I configure WebSPIRS for automatic login?

**A:**

Automatic login for WebSPIRS requires a change to a configuration file. Change to the `/usr/local/etc/webspirs/bin` directory on the web server. Use a text editor to edit the `webspirs.cfg` file and add these lines:

```
[SP]
username=<your ERL username>
password=<your ERL password>
```

For example:

```
[SP]
username=guest
password=guest
```

It is important to note that these three lines must be at the very beginning of the `webspirs.cfg` file. If they are not the first three lines in the file, then automatic login will not work. Additional details about configuring WebSPIRS can be found in Chapter 2, Installing and Configuring WebSPIRS.

---

**Q:**

Where can I get the WebSPIRS client software?

**A:**

The WebSPIRS ERL client software for the Linux platform is available on SilverPlatter's Software Resource CD. In addition, the software packages for both the Linux and Solaris platforms are located on SilverPlatter's Anonymous FTP server. The following instruction will help you **ftp** it:

1. Open an FTP connection to SilverPlatter's Anonymous FTP server (`ftp.silverplatter.com` or `192.80.71.12`). Login as **anonymous** or **ftp** and use your Internet e-mail address as the password.
2. Once you connect to the FTP server, change to the appropriate WebSPIRS client directory:

```
    /software/erl-clients/web/linux (for the Linux platform)
    /software/erl-clients/web/solaris (for the Solaris platform)
```
3. Set file transfer type to binary.
4. Copy the file from this subdirectory on the FTP server to your server's hard disk.
5. Exit out of FTP.

To install the software, see Chapter 2, Installing and Configuring WebSPIRS, or the instructions in the `install.txt` file.

**Q:**

What do the `cgibaby`, `cgichild`, and `cgiadult` executables do anyway?

**A:**

- **cgibaby**
- **cgichild** is the main WebSPIRS program that dispatches requests and runs all the time.
- **cgiadult** is a searching connection; you may see several of these.

When a user makes a request, the HTTP server launches `cgibaby`, which sends the request to `cgichild`. `cgichild` creates or uses one of the `cgiadult` connections to process the request. The request results are then sent from the `cgiadult` directly back to the `cgibaby`, and then to the HTTP server. A more detailed explanation of these programs can be found in Chapter 1, Overview.

---

**Q:**

Can WebSPIRS be used to access databases on a local area network?

**A:**

No, WebSPIRS has been developed as an ERL client. Because of this, it can only be used to access databases loaded on an ERL server. It cannot access databases located on non-ERL networks. Consult SilverPlatter for more information.

---

**Q:**

How do I eliminate a "Gateway not operating" message with a new WebSPIRS installation?

**A:**

This message means the gateway is not running. Run the following command to start it again:

---

**Q:**

What does the message "Gateway not found" mean?

**A:**

This message means that **cgibaby** could not find **cgichild**. The **cgichild** may not be running, or the `request_name` setting in the `webspirs.cfg` file and the `cgibaby.cfg` file do not match. Check the setting in those files. For information about the **cgibaby** and **cgichild** processes see Chapter 1, Overview.

---

**Q:**

I stopped the **cgichild** process and then tried to restart it again but could not. What's wrong?

**A:**

WebSPIRS communicates using TCP/IP connections. On some platforms shutting down **cgichild** leaves the pipes around for awhile, and they must time out. Wait a few minutes and try again.

---

**Q:**

Will WebSPIRS work with Netscape's web server software?

**A:**

Yes, however, the web server is configured through a menu system. Use the menu to set the HTTPD's user.

**Q:**

In what order does WebSPIRS list databases in the database selection screen?

**A:**

Databases are sorted on the ERL server not by WebSPIRS. The databases are listed in the order set by an ERL administrator through the `volsort` script.

---

**Q:**

Does WebSPIRS work with full text databases?

**A:**

Yes. Although WebSPIRS is unable to provide the user with a database-wide Table of Contents, the software does provide a Table of Contents for each record and is capable of supporting full-text databases.

---

**Q:**

What is the difference between the `webspirs.cgi` and `webspird.cgi` scripts?

**A:**

The `webspird.cgi` script is used to start and stop the WebSPIRS software. In addition, it provides status information on the program. To start WebSPIRS, change to the `/usr/local/etc/webspirs/bin` directory and enter:

```
./webspird.cgi start
```

To stop WebSPIRS, enter:

```
./webspird.cgi stop
```

To get information on the program's status, enter:

```
./webspird.cgi status
```

The `webspirs.cgi` script provides each WebSPIRS user with access to the ERL server. It is the script that implements the search software. See "The `.htaccess` File" section in Chapter 2, Installing and Configuring WebSPIRS.

---

**Q:**

How can the WebSPIRS software be configured to access a specific ERL server?

**A:**

On the WebSPIRS server, change to the `/usr/local/etc/webspirs/bin` directory. Using a text editor such as `vi`, edit the `erlclnt.cfg` file and set the `server_addr1` parameter to the correct ERL server address. For example:

```
server_addr1 = /2/erl2.silverplatter.com/416
```

Please note that there should only be one occurrence of the `server_addr1` parameter within this file.

**Q:**

How do users mark records in WebSPIRS?

**A:**

A checkbox appears before each record in the user's search results. The user can check records of interest and then redisplay them. The user can also print the records or save them to a file. The on-line help provides additional instructions for marking records.

---

**Q:**

Why does the `webspirs.pid` file have to be deleted after the web server is rebooted?

**A:**

The web server should not be rebooted while WebSPIRS is running. When it is, the `webspirs.pid` file is not deleted and restarting the software results in an error message. To prevent this from happening, stop the WebSPIRS software before rebooting the server. Change to the `/usr/local/etc/webspirs/bin` directory and enter:

```
./webspirs.cgi stop
```

---

**Q:**

How can unauthorized users be prevented from accessing WebSPIRS?

**A:**

The WebSPIRS 2.2 software, when used in conjunction with the ERL 2.1 server, does include the capability to prevent unauthorized users from accessing its web pages. Most web server software does include this type of capability. To prevent access, you can create a `.htaccess` file, as described in Chapter 2, *Installing and Configuring WebSPIRS*. In addition, the ERL server software provides security through IP address checking. Two files control IP address checking for TCP/IP clients. They are `ipincl` (IP Include) and `ipexcl` (IP Exclude). Please see the *ERL Administrator's Manual*

---

**Q:**

Can I search WebSPIRS directly with a URL?

**A:**

Yes, you can link to WebSPIRS and save time by searching directly with a URL. Here are two examples of how to do this.

The first example opens a specific database, skips the Database selection page, and goes to the Search page:

```
http://webspirs.silverplatter.com/cgi-bin/webspirs.cgi?sp.dbid,p=AESB  
&sp.nextform=search.htm&sp.search.term1,p=cash
```

The second example opens the MEDLINE databases it finds, skips the Database selection page, and goes to the Search page:

```
http://webspirs.silverplatter.com/cgi-bin/webspirs.cgi?sp.setid,p=ML&  
sp.nextform=search.htm&sp.search.term1,p=medication
```

Note that what follows `sp.search.term1.p` should be URL encoded. Blank spaces are particularly troublesome; for example, "jones in au" becomes "jones%20in%20au". See the `URLEncodeString` function of class `cgi_Config` in Chapter 5, *WebSPIRS Class Library*.



---

# Glossary

---

## API

**Application Program Interface.** A set of services which can be called by an application program, usually with a publicly specified interface.

## CGI script

**Common Gateway Interface script.** A platform-specific script which allows Web servers to interact with external processes. CGI programs can be written in a variety of languages which include C and perl.

## client

A platform-specific program responsible for interpreting the end user's queries and displaying the results. Sometimes referred to as the "user interface" or the "client application."

## client/server architecture

In a client/server architecture, the client functions for query formulation and display, and the server functions for query evaluation and display. The messaging system ties the client and server together.

## constructor

C++ member function with the same name as the class. It constructs objects of the class type and is invoked when the associated type is used in a definition.

## CORE

**Content Operative Retrieval Engine.** The CORE software is the essence of SilverPlatter's information search and retrieval technology. It is implemented using object-oriented techniques in the C programming language.

## CORE Wrapper

C++ classes that "wrap" and are dependent upon the CORE library, which was written in ANSI C.

## destructor

C++ member function whose name is the class name preceded by a "~" character. It destroys values of the class type.

**DTD**

**Document Type Definition.** The DTD is written using SGML and describes the sort of document to be used; for example, a manual or an article for a newspaper. The DTD contains declarations, each of which defines a construct to be used in a style of document. HTML is an SGML DTD.

**DXP**

**Data eXchange Protocol.** SilverPlatter's interface-independent retrieval protocol.

**ERL**

**Electronic Reference Library.** A multi-user application server implementation of SilverPlatter's CORE technology. The ERL retrieval engine runs on UNIX and can be accessed by clients over TCP/IP.

**explode**

A Boolean operation that calculates the OR (union) of a given word and all its narrower terms. If the end user chooses to explode a term, the selected term and all of its narrower terms are returned.

**find parser**

A CORE object that takes search syntax entered by the end user and converts it to search-engine-friendly syntax. It is an implementation of the **UI parser** object.

**form**

A form is a document having blank edit fields. The fields are filled in by and returned to the user. Some forms are simple, such as a query request form, and others are complex, such as an online registration form.

**FTP**

**File Transfer Protocol.** An industry-standard protocol used to transfer files to and from a remote computer.

**HTML**

**HyperText Markup Language.** A HTML document is an ASCII text file that contains embedded HTML tags. The tags are used to identify the structure of the document and to identify hyperlinks and their associated URLs.

**HTML templates**

Context-specific elements of the user's view of an ERL database, encoded with a macro language understood by WebSPIRS, and used as a starting point in the construction of the final HTML forms sent to the Web client.

**HTTP**

HyperText Transport Protocol.

**HTTPD**

HyperText Transport Protocol Daemon.

**Internet**

A world-wide computer network tying together educational institutions, businesses, governments, and individuals.

**IPX**

Internetworked Packet eXchange. Novell's protocol used by NetWare LANs. Layers of software above IPX can use IPX to deliver messages for a variety of LAN applications such as E-mail, databases, file services, and LAN printing.

**Local CORE**

CORE services implemented to access databases using direct reading of the local native file system.

**macro**

The term macro implies substitution. The SilverPlatter macros used in WebSPIRS are prewritten functions that provide for necessary chores, such as logging in and searching and retrieving information.

**MacTCP**

The software driver for the Macintosh operating system that implements the TCP/IP protocols. These protocols provide transmission services that are used by third-party applications such as electronic mail, remote login, file transfer, and database access.

**memory manager**

The API that provides services for managing memory while using the CORE. The memory manager manages two types of memory--addressable and virtual. It is sometimes abbreviated, "MM."

**NCSA**

National Center for Supercomputing Applications. Telnet software that implements TCP/IP protocols.

**NetWare**

NetWare is a family of local area network (LAN) operating systems from Novell, Inc., that run on IBM-compatible PCs, Macintosh computers, and Digital's VAX series.

**Network CORE**

CORE services implemented to access databases using a client/server protocol (DXP) interaction with a DXP-compatible application server.

**NFS**

**Network File System.** A network protocol developed and distributed by Sun Microsystems. NFS allows data to be shared among many users in a network, regardless of processor type, operating system, network architecture, or protocol.

**permuted list**

An ordered list of all the words that appear in thesaurus terms. Under each word is a list of the term(s) in which it appears.

**porting**

Converting software to run in a different computer environment.

**protocol**

A specific set of conventions for communications among computers.

**SGML**

**Standard Generalized Markup Language.** A standard for describing markup languages.

**SPIRS**

**SilverPlatter Information Retrieval System.** The following SPIRS applications have been created by SilverPlatter using the CORE:

- MacSPIRS is SP's retrieval system for the Macintosh
- PC-SPIRS is SP's first generation retrieval system for the IBM PC.
- UNIX-SPIRS is SP's retrieval system for the UNIX environment.
- WebSPIRS is SP's retrieval system for the Internet environment.
- WinSPIRS is SP's retrieval system for the Microsoft Windows environment.

**syntax**

A set of rules governing the structure of and relationship between symbols, words, and phrases in a language statement.

**TCP/IP**

**Transmission Control Protocol/Internet Protocol.** A communications protocol suite designed to interconnect a wide variety of computer equipment. TCP provides for the reliable transfer of data, while IP transmits the data through the network in the form of datagrams.

**template**

A HTML-formatted document that provides an outline of prepared HTML tags. Using a template saves the time spent inserting basic HTML tags and assures the correct order of the tags. Many WebSPIRS templates are encoded with a macro language.

**URL**

**Uniform Resource Locator.** Web browsers, such as Mosaic and Netscape, follow URLs to their source and display them.

**WILD**

**Worldwide Integrated Library of Databases.** The WILD Thing project supported different SGML DTDs.

**WWW**

**World Wide Web,** or Web for short. A method of sharing information on the Internet. The information is modeled as objects. An object can point to any other object on the Internet.

## A

access to WebSPIRS  
 restricting with the `.htaccess` file, 2-8  
`action.htm` template, 3-4, 3-11  
**AddAlert** function  
   `www_Alert` class, 5-12  
**AddArguments** function  
   `cgi_Config` class, 5-2  
**AddEnvironment** function  
   `cgi_Config` class, 5-2  
**AddExpandTag** function  
   `www_Arguments` class, 5-13  
**AddHelperConstructor** function  
   `www_HTML_Helper` class, 5-33  
**AddHttpTags** function  
   `cgi_Config` class, 5-2  
**AddTags** function  
   `cgi_Config` class, 5-3  
**AlwaysGenerateHidden** function  
   `www_Macro` class, 5-35  
**AppendCfg** function  
   `cgi_Config` class, 5-3  
**Assert** function  
   `www_Environment` class, 5-18  
**AssignValue** function  
   `www_User` class, 5-49

## B

**BadUser** function  
   `www_ERLConnection` class, 5-22  
**BetweenTOC** function  
   `www_Record` class, 5-40  
 Blat utility, 2-7  
**BuildAllFieldSets** function  
   `www_Database` class, 5-17  
 building WebSPIRS, 4-25  
**BuildSGMLToCURL** function  
   `www_Wild` class, 5-51  
 bypassing the Login page, 3-4

## C

C++ implementation, 4-25  
 coding macro functions, 4-27  
`cgi_Config` class  
   `~cgi_config` destructor, 5-2  
   **AddArguments** function, 5-2  
   **AddEnvironment** function, 5-2  
   **AddHttpTags** function, 5-2  
   **AddTags** function, 5-3  
   **AppendCfg** function, 5-3  
   `cgi_Config` constructors, 5-2  
   **ConvertToHttp** function, 5-3  
   **Dump** function, 5-3  
   **MergeItem** function, 5-3

**operator=** function, 5-2  
   **ReadRequest** function, 5-3  
   **RestoreCodedCharacters** function, 5-4  
   **RoleInDefaults** function, 5-3  
   **URLEncodeString** function, 5-3  
`cgiadult` process, 1-2  
`cgibaby` process, 1-2  
`cgichild` process, 1-2  
 changing the number of records displayed, 3-6  
 changing the template title, 3-6  
`checkbox.htm` template, 3-11  
 classes, 5-1  
   `cgi_Config`, 5-2  
   `dxp_to_html`, 5-5  
   `erlAdmin`, 5-6  
   `erlAdmin_DBInfo`, 5-7  
   `erlAdmin_UserInfo`, 5-8  
   hierarchical drawing, 5-1  
   `sgml_Field`, 5-10  
   `www_Admin`, 5-11  
   `www_Alert`, 5-12  
   `www_Arguments`, 5-13  
   `www_Database`, 5-15  
   `www_Environment`, 5-18  
   `www_ERLConnection`, 5-20  
   `www_Field`, 5-24  
   `www_FSI`, 5-25  
   `www_Guide`, 5-27  
   `www_HTML_Helper`, 5-28  
   `www_Macro`, 5-34  
   `www_MacroCaller`, 5-36  
   `www_Record`, 5-37  
   `www_Request`, 5-41  
   `www_Search`, 5-44  
   `www_Server`, 5-46  
   `www_Template`, 5-48  
   `www_User`, 5-49  
   `www_Wild`, 5-51  
`clrsrch.htm` template, 3-11  
 configuring WebSPIRS  
   activating mail, 2-6  
   `cgibaby.cfg` file, 2-6  
   changing the socket number, 2-6  
   creating a mail script, 2-6  
   creating the `.htaccess` file, 2-8  
   `erlclnt.cfg` file, 2-7  
   `mime.types` file, 2-8  
   restricting access, 2-8  
   setting a debug value, 2-6  
   setting the connections, 2-6  
   `webspirs.cfg` file, 2-6  
**ConnectionDied** function  
   `www_ERLConnection` class, 5-23  
**ConstructHelpers** function  
   `www_HTML_Helper` class, 5-33  
 controlling the flow of pages  
   using a Submit button, 3-8  
   using a URL, 3-8

**Convert** function  
  **dxp\_to\_html** class, 5-5  
**ConvertConcatSymbol** function  
  **www\_Request** class, 5-43  
**ConvertToHttp** function  
  **cgi\_Config** class, 5-3  
**CopyToVariable** function  
  **www\_FSI** class, 5-26  
customizing templates, 3-1  
  bypassing the Login page, 3-4  
  changing the background graphic, 3-7  
  changing the graphical interface, 3-6  
  changing the logo graphic, 3-7  
  changing the records display default number, 3-6  
  changing the template title, 3-6  
  changing the toolbar, 3-6  
  controlling the flow of pages, 3-8  
  creating a table, 3-7  
  displaying specific fields, 3-5  
  kinds of customizations, 3-4  
  preselecting databases, 3-5

## D

**database.htm** template, 3-9  
**DatabaseForEach** function  
  **www\_Database** class, 5-16  
**dbitem.htm** template, 3-11  
**dblitmck.htm** template, 3-11  
**DbIRefresh** function  
  **www\_Request** class, 5-43  
debugging WebSPIRS, 1-3  
displaying specific fields, 3-5  
**DoCommands** function  
  **www\_Request** class, 5-42  
**DoFSIToc** function  
  **www\_FSI** class, 5-25  
**DoPermutedList** function  
  **www\_Search** class, 5-45  
**DoURL** function  
  **www\_Record** class, 5-39  
**Dump** function  
  **cgi\_Config** class, 5-3  
**dxp\_to\_html** class  
  ~**dxp\_to\_html** destructor, 5-5  
  **Convert** function, 5-5  
  **dxp\_to\_html** constructor, 5-5  
  **SetDatabase** function, 5-5  
  **SetRequest** function, 5-5  
  **SetTranslatefromDos** function, 5-5  
**DXPError** function  
  **www\_ERLConnection** class, 5-23

## E

EMWAC server, 2-1  
**erlAdmin** class  
  ~**erlAdmin** destructor, 5-6  
  **erlAdmin** constructor, 5-6

**erlAdmin\_DBInfo** class  
  **erlAdmin\_DBInfo** constructor, 5-7  
  **erlAdmin\_DBInfo** destructor, 5-7  
  **GetCostPerAbstract** function, 5-7  
  **GetCostPerRecord** function, 5-7  
  **SetDatabase** function, 5-7  
**erlAdmin\_UserInfo** class  
  ~**erlAdmin\_UserInfo** destructor, 5-8  
  **erlAdmin\_UserInfo** constructor, 5-8  
  **GetMaxCharge** function, 5-9  
  **GetNumberCurrentlyLogged** function, 5-9  
  **GetTotalCharge** function, 5-9  
  **GetTotalLogins** function, 5-9  
  **GetUserId** function, 5-9  
  **GetUserName** function, 5-8  
  **SetServer** function, 5-8  
  **SetUserName** function, 5-8  
**EvaluateAlert** function  
  **www\_Alert** class, 5-12  
examples  
  adding a macro using C++, 4-26  
  implementing a **www\_HTML\_Helper** class, 4-26  
**ExpansionMethod** function  
  **www\_Macro** class, 5-35  
**exphtml.htm** template, 3-11  
**expmail.htm** template, 3-11  
**exprow.htm** template, 3-11  
**expterm.htm** template, 3-11

## F

**FatalErrorOccurred** function  
  **www\_ERLConnection** class, 5-20  
**fielditm.htm** template, 3-11  
fields  
  changing the default display fields, 3-5  
**foot.htm** template, 3-11  
**ForEach** function  
  **www\_User** class, 5-50  
FORMs in WebSPIRS, 3-4  
frequently asked questions  
  Can I search WebSPIRS directly with a URL?, 6-5  
  Can the WebSPIRS interface be changed?, 6-1  
  Can WebSPIRS be run on the same server as the  
    ERL server?, 6-1  
  Can WebSPIRS be used to access databases on a  
    LAN?, 6-3  
  Does WebSPIRS work with full text databases?,  
    6-4  
  How can I configure WebSPIRS for automatic  
    login?, 6-2  
  How can the WebSPIRS software be configured to  
    access a specific ERL server?, 6-4  
  How can unauthorized users be prevented from  
    accessing WebSPIRS?, 6-5  
  How do I eliminate a "Gateway not operating"  
    message?, 6-3  
  How do users mark records in WebSPIRS?, 6-5

I stopped the cgichild process and cannot start it again. What's wrong?, 6-3  
 In what order does WebSPIRS list databases?, 6-4  
 What are the minimum requirements for WebSPIRS?, 6-1  
 What do the cgibaby, cgichild, and cgiadult executables do anyway?, 6-3  
 What does the message "Gateway not found" mean?, 6-3  
 What is the difference between the `webspirs.cgi` and `webspird.cgi` scripts?, 6-4  
 What is the WebSPIRS software?, 6-1  
 Where can I get the WebSPIRS client software?, 6-2  
 Why does the `webspirs.pid` file have to be deleted after the web server is rebooted?, 6-5  
 Will WebSPIRS work with Netscape's web server software?, 6-3  
**FSIToSearch** function  
   **www\_FSI** class, 5-25  
 ftp site, 2-1

## G

**GenerateForm** function  
   **www\_Request** class, 5-42  
**GenerateURL** function  
   **www\_User** class, 5-50  
**GetAbstractCostPerRecord** function  
   **www\_Record** class, 5-40  
**GetASLList** function  
   **www\_Search** class, 5-44  
**GetCitation** function  
   **www\_Wild** class, 5-51  
**GetClientAddress** function  
   **www\_ERLConnection** class, 5-21  
**GetContentField** function  
   **sgml\_Field** class, 5-10  
**GetContentType** function  
   **sgml\_Field** class, 5-10  
**GetCostPerAbstract** function  
   **erlAdmin\_DBInfo** class, 5-7  
   **www\_Admin** class, 5-11  
**GetCostPerRecord** function  
   **erlAdmin\_DBInfo** class, 5-7  
**GetCurrentDatabaseList** function  
   **www\_Environment** class, 5-19  
**GetCurrentDBID** function  
   **www\_Record** class, 5-39  
**GetCurrentDBName** function  
   **www\_Record** class, 5-40  
**GetCurrentPlainText** function  
   **www\_Record** class, 5-39  
**GetCurrentRecordText** function  
   **www\_Record** class, 5-39  
**GetCurrentRecordURL** function  
   **www\_Record** class, 5-40  
**GetCurrentRecordValue** function

**www\_Record** class, 5-39  
**GetDatabase** function  
   **www\_Database** class, 5-15  
   **www\_HTML\_Helper** class, 5-30  
   **www\_Request** class, 5-43  
**GetDatabaseDescriptions** function  
   **www\_Database** class, 5-16  
**GetDatabaseList** function  
   **www\_Environment** class, 5-19  
**GetDatabaseName** function  
   **www\_Database** class, 5-15  
**GetDatabaseTag** function  
   **www\_Database** class, 5-16  
**GetDatabaseTitleScreens** function  
   **www\_Database** class, 5-16  
**GetDatesCovered** function  
   **www\_Database** class, 5-16  
**GetDescription** function  
   **www\_HTML\_Helper** class, 5-31  
   **www\_Macro** class, 5-35  
**GetEchoFormsFlag** function  
   **www\_Server** class, 5-47  
**GetEchoRequestsFlag** function  
   **www\_Server** class, 5-46  
**GetERLPath** function  
   **www\_Environment** class, 5-19  
**GetErrorMessage** function  
   **www\_ERLConnection** class, 5-21  
   **www\_Request** class, 5-43  
**GetExpansionMethod** function  
   **www\_Macro** class, 5-35  
**GetExpansionString** function  
   **www\_HTML\_Helper** class, 5-30  
**GetExpiredServer** function  
   **www\_ERLConnection** class, 5-21  
**GetFieldList** function  
   **www\_Field** class, 5-24  
**GetFieldListForEach** function  
   **www\_Field** class, 5-24  
**GetForm** function  
   **www\_Template** class, 5-49  
**GetFSIList** function  
   **www\_FSI** class, 5-25  
**GetGuideDBName** function  
   **www\_Guide** class, 5-27  
**GetHelper** function  
   **www\_Request** class, 5-43  
**GetHelperCount** function  
   **www\_HTML\_Helper** class, 5-33  
   **www\_Request** class, 5-43  
**GetHiddenVariables** function  
   **www\_HTML\_Helper** class, 5-30  
   **www\_Request** class, 5-42  
**GetHowMany** function  
   **www\_Record** class, 5-38  
**GetKeyFoundFlag** function  
   **www\_HTML\_Helper** class, 5-29  
**GetLastURL** function



- www\_Record** class, 5-40
- GetListForEach** function
  - www\_Database** class, 5-16
- GetMacro** function
  - www\_HTML\_Helper** class, 5-30
- GetMacroCount** function
  - www\_HTML\_Helper** class, 5-30
- GetMacroList** function
  - www\_User** class, 5-49
- GetMaxCharge** function
  - erlAdmin\_UserInfo** class, 5-9
- GetName** function
  - www\_HTML\_Helper** class, 5-28
  - www\_Macro** class, 5-34
- GetNotation** function
  - www\_Wild** class, 5-51
- GetNumberCurrentlyLogged** function
  - erlAdmin\_UserInfo** class, 5-9
- GetPtr** function
  - www\_Environment** class, 5-19
- GetRecordCounts** function
  - www\_Record** class, 5-37
- GetRecordField** function
  - www\_Record** class, 5-40
- GetRecordFieldText** function
  - www\_Record** class, 5-38
- GetRecordNumber** function
  - www\_Record** class, 5-38
- GetRecords** function
  - www\_HTML\_Helper** class, 5-29
  - www\_Record** class, 5-38
  - www\_Request** class, 5-43
  - www\_Search** class, 5-44
- GetRecordsExtracted** function
  - www\_Record** class, 5-37
- GetRecordText** function
  - www\_Record** class, 5-37
- GetRequest** function
  - www\_HTML\_Helper** class, 5-30
- GetSearchText** function
  - www\_Search** class, 5-45
- GetSelf** function
  - www\_ERLConnection** class, 5-22
- GetTableOfContents** function
  - www\_Record** class, 5-38
- GetTermDefinition** function
  - www\_Search** class, 5-45
- GetTotalCharge** function
  - erlAdmin\_UserInfo** class, 5-9
- GetTotalLogins** function
  - erlAdmin\_UserInfo** class, 5-9
- GetUserId** function
  - erlAdmin\_UserInfo** class, 5-9
- GetUserName** function
  - erlAdmin\_UserInfo** class, 5-8
- GetValue** function
  - www\_User** class, 5-50
- Go** function

- www\_Server** class, 5-46

## H

- head.htm template, 3-11
- hierarchical class drawing, 5-1
- hotlink.htm template, 3-9

## I

- Idle** function
  - www\_Environment** class, 5-18
- IfCond** function
  - www\_User** class, 5-50
- implementing WebSPIRS
  - C++ language, 4-25
- IncludeTemplate** function
  - www\_User** class, 5-49
- index.htm template, 3-9
- indterm.htm template, 3-11
- Init** function
  - www\_HTML\_Helper** class, 5-28
- Initialize** function
  - www\_HTML\_Helper** class, 5-32
- installing WebSPIRS, 2-1
  - downloading a browser, 2-1
  - downloading a free server, 2-1
  - downloading the Apache server, 2-1
  - downloading the EMWAC server, 2-1
  - downloading the package file, 2-1
  - downloading the WebSite server, 2-1
  - Linux platform, 2-2
  - preparing to install, 2-1
  - saving changes before you install, 2-1
  - Solaris platform, 2-3
  - Windows NT platform, 2-5
- internal processes
  - cgadult**, 1-2
  - cgibaby**, 1-1
  - cgichild**, 1-2
- InterpretArguments** function
  - www\_HTML\_Helper** class, 5-31
- IsContextVariable** function
  - www\_Macro** class, 5-35
- IsTemplateDriven** function
  - www\_Request** class, 5-42

## L

- Linux platform
  - installation procedure, 2-2
  - installing manually, 2-2
  - installing with pkgtool, 2-2
  - minimum hardware requirements, 2-2
- LoadGatewayFile** function
  - www\_HTML\_Helper** class, 5-32
- log files
  - wwwbaby.req**, 1-3
  - wwwForm.log**, 1-4

- wwwlast.req, 1-4
  - webreq.log, 1-4
  - webuser.log, 1-4
  - Login** function
    - www\_ERLConnection class, 5-22
  - login.htm template, 3-9
  - LoginFailed** function
    - www\_ERLConnection class, 5-20
  - LoginRequired** function
    - www\_HTML\_Helper class, 5-30
    - www\_Macro class, 5-35
  - logout.htm template, 3-9
  - Lookup** function
    - www\_HTML\_Helper class, 5-29
    - www\_MacroCaller class, 5-36
- M**
- macro commands, 4-1
    - sp.admin.balance, 4-6
    - sp.admin.total.used, 4-6
    - sp.asl.list, 4-14
    - sp.assign, 4-21
    - sp.avail.dbs.foreach, 4-7
    - sp.checked, 4-20
    - sp.currentrecord.absolute.url, 4-9
    - sp.currentrecord.abstract.cost, 4-9
    - sp.erl.logout, 4-6
    - sp.erl.message.of.the.day, 4-6
    - sp.expansionpart, 4-17
    - sp.field.list, 4-16
    - sp.foreach, 4-22
    - sp.fsi.copytovariable, 4-15
    - sp.fsi.list, 4-15
    - sp.fsi.tosearch, 4-14
    - sp.generate\_url, 4-23
    - sp.guide.toc, 4-19
    - sp.guide.topic, 4-19
    - sp.if, 4-23
    - sp.include, 4-23
    - sp.isinlist, 4-20
    - sp.makespurl, 4-10
    - sp.opened.dbs.foreach, 4-7
    - sp.record.counts, 4-10
    - sp.record.initialize, 4-10
    - sp.record.text, 4-10
    - sp.record.toc, 4-9
    - sp.searchhistory.build, 4-13
    - sp.term.definition, 4-17
    - sp.term.prepare\_detail, 4-17
    - sp.url.p, 4-10
    - sp.webspirs.version, 4-6
  - macro variables, 4-1
    - sp.age.subheading, 4-18
    - sp.avail.dbs.criteria.p, 4-9
    - sp.avail.dbs.item.endindent, 4-8
    - sp.avail.dbs.item.id, 4-8
    - sp.avail.dbs.item.indent, 4-8
    - sp.avail.dbs.item.name, 4-8
    - sp.back.form.p, 4-4
    - sp.back.form.title.p, 4-4
    - sp.check.name, 4-20
    - sp.check.selections, 4-21
    - sp.currentrecord, 4-11
    - sp.currentrecord.dbname, 4-11
    - sp.dbid.p, 4-8
    - sp.erl.server.address, 4-7
    - sp.export.mailto.p, 4-5
    - sp.export.range.p, 4-4
    - sp.export.recnums.p, 4-5
    - sp.export.save.history.p, 4-5
    - sp.form.foot.p, 4-4
    - sp.form.head.p, 4-3
    - sp.form.search.p, 4-3
    - sp.form.show.p, 4-3
    - sp.form.top.p, 4-4
    - sp.fsi.fields.p, 4-15
    - sp.fsi.howmany.p, 4-16
    - sp.fsi.term.p, 4-16
    - sp.guide.dbname, 4-19
    - sp.hotlink.form.p, 4-13
    - sp.login\_error, 4-3
    - sp.mailcmd, 4-24
    - sp.nextform, 4-3
    - sp.opened.dbs.item.name, 4-8
    - sp.output, 4-24
    - sp.password, 4-2
    - sp.perm.word.p, 4-18
    - sp.record.fields.p, 4-11
    - sp.record.howmany.p, 4-11
    - sp.record.labels.p, 4-12
    - sp.record.lastshown.p, 4-12
    - sp.record.marked.pp, 4-21
    - sp.record.number.p, 4-11
    - sp.record.sortfields.p, 4-12
    - sp.record.sortlimit.p, 4-12
    - sp.record.sortrecords.p, 4-12
    - sp.record.source.p, 4-11
    - sp.search.invalid\_message, 4-3
    - sp.search.value.p, 4-13
    - sp.searchhistory.operator, 4-13
    - sp.select.terms.pp, 4-18
    - sp.tbar.page.value, 4-4
    - sp.template\_description, 4-3
    - sp.term.narrower\_terms, 4-18
    - sp.term.related\_terms, 4-18
    - sp.thesaurus.term.p, 4-19
    - sp.thisform, 4-2
    - sp.topical.subheading, 4-18
    - sp.username, 4-2
    - sp.webspirs.docdir, 4-7
  - macros
    - adding new macros, 4-24
    - administration, 4-6
    - automatic subject lookup (ASL), 4-14
    - categories, 4-1
    - example, 4-1
    - field list, 4-16
    - field-specific index (FSI), 4-14

- general database, 4-7
- guide keyword, 4-19
- interface-specific variables, 4-2
- marked records, 4-20
- miscellaneous, 4-21
- nested, 4-2
- search, 4-13
- text display, 4-9
- thesaurus term, 4-16
- useful services, 4-28

mail activation, 2-6

- Linux platform, 2-6
- Windows NT platform, 2-7

maildone.htm template, 3-11

mailopts.htm template, 3-10

**MakeCheckboxHTML** functions

- www\_HTML\_Helper class, 5-31

**MakeGuideToc** function

- www\_Guide class, 5-27

**MakeGuideTopic** function

- www\_Guide class, 5-27

**MakeHyperLink** function

- www\_HTML\_Helper class, 5-32

**MakeLink** function

- www\_Wild class, 5-51

**MakeOptionHTML** functions

- www\_HTML\_Helper class, 5-31

**MakeSPUrl** function

- www\_Record class, 5-40

**MakeUrl** function

- www\_HTML\_Helper class, 5-32

**MakeWinspirsLink** function

- www\_Database class, 5-16

**MaxUsers** function

- www\_ERLConnection class, 5-22

**MergeItem** function

- cgi\_Config class, 5-3

mime.types file, 2-8

motd.htm template, 3-10

mrkclear.htm template, 3-12

## N

Netscape browser, 2-1

news.htm template, 3-12

## O

opening WebSPIRS on your browser, 2-8

overview of WebSPIRS, 1-1

## P

pagesize.htm template, 3-12

password.htm template, 3-10

**PasswordExpired** function

- www\_ERLConnection class, 5-21, 5-22

**PrepareTerm** function

- www\_Search class, 5-45

preparing to install WebSPIRS, 2-1

**PreprocessRequest** function

- www\_FSI class, 5-26
- www\_HTML\_Helper class, 5-29
- www\_Record class, 5-38

preselecting databases, 3-5

prntopts.htm template, 3-10

**ProtocolError** function

- www\_ERLConnection class, 5-23

## R

**ReadRequest** function

- cgi\_Config class, 5-3

recdtoc.htm template, 3-10

recfmt.htm template, 3-12

recfmtck.htm template, 3-12

recfmttrw.htm template, 3-12

rechits.htm template, 3-11

**RecordInitialize** function

- www\_Record class, 5-38

reprint.htm template, 3-10

recsmark.htm template, 3-10

**Refresh** function

- www\_ERLConnection class, 5-20

**RemoveDatabaseList** function

- www\_Environment class, 5-19

**Reset** function

- www\_Alert class, 5-12
- www\_Database class, 5-15
- www\_FSI class, 5-25
- www\_HTML\_Helper class, 5-30
- www\_Record class, 5-37
- www\_Request class, 5-42
- www\_Search class, 5-44

**RestoreCodedCharacters** function

- cgi\_Config class, 5-4

restricting access to WebSPIRS, 2-8

**RoleInDefaults** function

- cgi\_Config class, 5-3

**RunMacro** function

- www\_Request class, 5-42

## S

saveopts.htm template, 3-10

scroll.htm template, 3-10

search.htm template, 3-10

seldbs.htm template, 3-11

**SetClientAddress** function

- www\_ERLConnection class, 5-20

**SetConfirmPassword** function

- www\_ERLConnection class, 5-21

**SetCount** function

- www\_HTML\_Helper class, 5-33

**SetDatabase** function

- dxp\_to\_html class, 5-5
- erlAdmin\_DBInfo class, 5-7

**SetDescription** function

- www\_HTML\_Helper** class, 5-31
  - www\_Macro** class, 5-35
- SetEchoFormsFlag** function
  - www\_Server** class, 5-46, 5-47
- SetErrorMessage** function
  - www\_ERLConnection** class, 5-21
- SetExpansionMethod** function
  - www\_Macro** class, 5-35
- setfield.htm** template, 3-10
- SetFieldName** function
  - sgml\_Field** class, 5-10
- SetFileNames** function
  - www\_Server** class, 5-47
- SetFormContents** function
  - www\_Template** class, 5-48
- SetKeyFoundFlag** function
  - www\_HTML\_Helper** class, 5-29
- SetLoginRequired** function
  - www\_Macro** class, 5-35
- SetMacro** function
  - www\_HTML\_Helper** class, 5-33
- SetName** function
  - www\_HTML\_Helper** class, 5-29
- SetNewPassword** function
  - www\_ERLConnection** class, 5-21
- SetPipeName** function
  - www\_Server** class, 5-47
- SetRequest** function
  - dxp\_to\_html** class, 5-5
  - www\_Environment** class, 5-19
  - www\_ERLConnection** class, 5-21
  - www\_Request** class, 5-42
- SetServer** function
  - erlAdmin\_UserInfo** class, 5-8
- SetTranslateFromDos** function
  - dxp\_to\_html** class, 5-5
- SetUserName** function
  - erlAdmin\_UserInfo** class, 5-8
- sgml\_Field** class
  - ~sgml\_Field** destructor, 5-10
  - GetContentField** function, 5-10
  - GetContentType** function, 5-10
  - SetFieldName** function, 5-10
  - sgml\_Field** constructors, 5-10
- showlrec.htm** template, 3-10
- ShowMessage** function
  - www\_ERLConnection** class, 5-22
- Solaris platform
  - installation procedure, 2-3
  - minimum hardware requirements, 2-3
- SortRecords** function
  - www\_Record** class, 5-38
- sp.admin.balance** macro command, 4-6
- sp.admin.total.used** macro command, 4-6
- sp.age.subheading** macro variable, 4-18
- sp.asl.list** macro command, 4-14
- sp.assign** macro command, 4-21
- sp.avail.dbs.criteria.p** macro variable, 4-9
- sp.avail.dbs.foreach** macro command, 4-7
- sp.avail.dbs.item.endindent** macro variable, 4-8
- sp.avail.dbs.item.id** macro variable, 4-8
- sp.avail.dbs.item.indent** macro variable, 4-8
- sp.avail.dbs.item.name** macro variable, 4-8
- sp.back.form.p** macro variable, 4-4
- sp.back.form.title.p** macro variable, 4-4
- sp.check.name** macro variables, 4-20
- sp.check.selections** macro variable, 4-21
- sp.checked** macro command, 4-20
- sp.currentrecord** macro variable, 4-11
- sp.currentrecord.absolute.url** macro command, 4-9
- sp.currentrecord.abstract.cost** macro command, 4-9
- sp.currentrecord.dbname** macro variable, 4-11
- sp.dbid.p** macro variable, 4-8
- sp.erl.logout** macro command, 4-6
- sp.erl.message.of.the.day** macro command, 4-6
- sp.erl.server.address** macro variable, 4-7
- sp.expansionpart** macro command, 4-17
- sp.export.mailto.p** macro variable, 4-5
- sp.export.range.p** macro variable, 4-4
- sp.export.recnums.p** macro variables, 4-5
- sp.export.save.history.p** macro variable, 4-5
- sp.field.list** macro command, 4-16
- sp.foreach** macro command, 4-22
- sp.form.foot.p** macro variable, 4-4
- sp.form.head.p** macro variable, 4-3
- sp.form.search.p** macro variable, 4-3
- sp.form.show.p** macro variable, 4-3
- sp.form.top.p** macro variable, 4-4
- sp.fsi.copypvariable** macro command, 4-15
- sp.fsi.fields.p** macro variable, 4-15
- sp.fsi.howmany.p** macro variable, 4-16
- sp.fsi.list** macro command, 4-15
- sp.fsi.term.p** macro variable, 4-16
- sp.fsi.tosearch** macro command, 4-14
- sp.generate\_url** macro command, 3-8, 4-23
- sp.guide.dbname** macro variable, 4-19
- sp.guide.toc** macro command, 4-19
- sp.guide.topic** macro command, 4-19
- sp.hotlink.form.p** macro variable, 4-13
- sp.if** macro command, 4-23
- sp.include** macro command, 4-23
- sp.isinlist** macro command, 4-20
- sp.login\_error** macro variable, 4-3
- sp.mailcmd** macro variable, 4-24
- sp.makespurl** macro command, 4-10
- sp.nextform** macro variable, 4-3
- sp.opened.dbs.foreach** macro command, 4-7

sp.opened.dbs.item.name  
4-8

sp.output macro variable, 4-24

sp.password macro variable, 4-2

sp.perm.word.p macro variable, 4-18

sp.record.counts macro command, 4-10

sp.record.fields.p macro variable, 4-11

sp.record.howmany.p macro variable, 4-11

sp.record.initialize macro command, 4-10

sp.record.labels.p macro variable, 4-12

sp.record.lastshown.p macro variable, 4-12

sp.record.marked.pp macro variable, 4-21

sp.record.number.p macro variable, 4-11

sp.record.sortfields.p macro variable,  
4-12

sp.record.sortlimit.p macro variable, 4-12

sp.record.sortrecords.p macro variable,  
4-12

sp.record.source.p macro variable, 4-11

sp.record.text macro command, 4-10

sp.record.toc macro command, 4-9

sp.search.invalid\_message macro variable,  
4-3

sp.search.value.p macro variable, 4-13

sp.searchhistory.build macro command,  
4-13

sp.searchhistory.operator macro variable,  
4-13

sp.select.terms.pp macro variable, 4-18

sp.tbar.page.value macro variable, 4-4

sp.template\_description macro variable,  
3-6, 4-3

sp.term.definition macro command, 4-17

sp.term.narrower\_terms macro variable,  
4-18

sp.term.prepare\_detail macro command,  
4-17

sp.term.related\_terms macro variable, 4-18

sp.thesaurus.term.p macro variable, 4-19

sp.thisform macro variable, 4-2

sp.topical.subheading macro variable, 4-18

sp.url.p macro command, 4-10

sp.username macro variable, 4-2

sp.webspirs.docdir macro variable, 4-7

sp.webspirs.version macro command, 4-6

srchcomn.htm template, 3-11

srchterm.htm template, 3-12

**Start** function

- www\_Environment** class, 5-18

subject.htm template, 3-10

subtermd.htm template, 3-12

suggest.htm template, 3-10

sugsrch.htm template, 3-12

sugterm.htm template, 3-12

## T

table of helpers, 5-32

tables

creating, 3-7

example, 3-8

tags

- [sp\_block]...[/sp\_block], 4-2
- [SP\_MACRO]...[/SP\_MACRO], 4-1

tbar.htm template, 3-12

templates

- action.htm, 3-11
- checkbox.htm, 3-11
- clrsrch.htm, 3-11
- complete pages, 3-9
- customizing, 3-1
- database.htm, 3-9
- dbitem.htm, 3-11
- dblitmck.htm, 3-11
- described, 3-1
- diagram of main templates, 3-9
- editing, 3-2
- encoded macros, 3-1
- exphtml.htm, 3-11
- expmail.htm, 3-11
- exprow.htm, 3-11
- expterm.htm, 3-11
- fielditm.htm, 3-11
- foot.htm, 3-11
- head.htm, 3-11
- hotlink.htm, 3-9
- index.htm, 3-9
- indterm.htm, 3-11
- large fragments, 3-11
- location, 3-2
- login.htm, 3-9
- logout.htm, 3-9
- maildone.htm, 3-11
- mailopts.htm, 3-10
- motd.htm, 3-10
- mrkclear.htm, 3-12
- news.htm, 3-12
- pagesize.htm, 3-12
- password.htm, 3-10
- prntopts.htm, 3-10
- recdtoc.htm, 3-10
- recfmt.htm, 3-12
- recfmtck.htm, 3-12
- recfmtrw.htm, 3-12
- rechits.htm, 3-11
- reprint.htm, 3-10
- recsmark.htm, 3-10
- saveopts.htm, 3-10
- scroll.htm, 3-10
- search.htm, 3-10
- seldbs.htm, 3-11
- setfield.htm, 3-10
- showlrec.htm, 3-10
- srchcomn.htm, 3-11
- srchterm.htm, 3-12

states

- editable, 3-2
- processed, 3-3

subject.htm, 3-10  
 subtermd.htm, 3-12  
 suggest.htm, 3-10  
 sugsrch.htm, 3-12  
 sugterm.htm, 3-12  
 tbar.htm, 3-12  
 thesite.htm, 3-12  
 thesterm.htm, 3-10  
 top.htm, 3-12  
 urlsrch.htm, 3-10  
 utility fragments, 3-11  
**Terminate** function  
   **www\_HTML\_Helper** class, 5-32  
 thesite.htm template, 3-12  
 thesterm.htm template, 3-10  
 top.htm template, 3-12  
 tutorial for customizing WebSPIRS templates, 3-1

## U

**URLDecodeString** function  
   **cgi\_Config** class, 5-4  
**URLEncodeString** function  
   **cgi\_Config** class, 5-3  
 URLs  
   Library of Congress graphics resources, 3-6  
   Library of Congress Internet resources, 3-2  
 urlsrch.htm template, 3-10

## W

WebSite server, 2-1  
 WebSPIRS  
   major features, 1-1  
   overview, 1-1  
   process, 1-3  
   process diagram, 1-1  
 Windows NT platform  
   installation procedure, 2-5  
   minimum hardware requirements, 2-5  
**www\_Admin** class  
   ~**www\_Admin** destructor, 5-11  
   **GetCostPerAbstract** function, 5-11  
   **www\_Admin** constructor, 5-11  
**www\_Alert** class  
   ~**www\_Alert** destructor, 5-12  
   **AddAlert** function, 5-12  
   **EvaluateAlert** function, 5-12  
   **Reset** function, 5-12  
   **www\_Alert** constructor, 5-12  
**www\_Arguments** class  
   ~**www\_Arguments** destructor, 5-13  
   **AddExpandTag** function, 5-13  
   **operator=** function, 5-13  
   **www\_Arguments** constructors, 5-13  
**www\_Database** class  
   ~**www\_Database** destructor, 5-15  
   **BuildAllFieldSets** function, 5-17  
   **DatabaseForEach** function, 5-16

**GetDatabase** function, 5-15  
   **GetDatabaseDescriptions** function, 5-16  
   **GetDatabaseName** function, 5-15  
   **GetDatabaseTag** function, 5-16  
   **GetDatabaseTitleScreens** function, 5-16  
   **GetDatesCovered** function, 5-16  
   **GetListForEach** function, 5-16  
   **MakeWinspirsLink** function, 5-16  
   **Reset** function, 5-15  
   **www\_Database** constructor, 5-15  
**www\_Environment** class  
   ~**www\_Environment** destructor, 5-18  
   **Assert** function, 5-18  
   **GetCurrentDatabaseList** function, 5-19  
   **GetDatabaseList** function, 5-19  
   **GetERLPath** function, 5-19  
   **GetPtr** function, 5-19  
   **Idle** function, 5-18  
   **RemoveDatabaseList** function, 5-19  
   **SetRequest** function, 5-19  
   **Start** function, 5-18  
   **www\_Environment** constructor, 5-18  
**www\_ERLConnection** class  
   ~**www\_ERLConnection** destructor, 5-20  
   **BadUser** function, 5-22  
   **ConnectionDied** function, 5-23  
   **DXPError** function, 5-23  
   **FatalErrorOccurred** function, 5-20  
   **GetClientAddress** function, 5-21  
   **GetErrorMessage** function, 5-21  
   **GetExpiredServer** function, 5-21  
   **GetSelf** function, 5-22  
   **Login** function, 5-22  
   **LoginFailed** function, 5-20  
   **MaxUsers** function, 5-22  
   **PasswordExpired** function, 5-21, 5-22  
   **ProtocolError** function, 5-23  
   **Refresh** function, 5-20  
   **SetClientAddress** function, 5-20  
   **SetConfirmPassword** function, 5-21  
   **SetErrorMessage** function, 5-21  
   **SetNewPassword** function, 5-21  
   **SetRequest** function, 5-21  
   **ShowMessage** function, 5-22  
   **www\_ERLConnection** constructor, 5-20  
**www\_Field** class  
   ~**www\_Field** destructor, 5-24  
   **GetFieldList** function, 5-24  
   **GetFieldListForEach** function, 5-24  
   **www\_Field** constructor, 5-24  
**www\_FSI** class  
   ~**www\_FSI** destructor, 5-25  
   **CopyToVariable** function, 5-26  
   **DoFSIToc** function, 5-25  
   **FSIToSearch** function, 5-25  
   **GetFSIList** function, 5-25  
   **PreprocessRequest** function, 5-26  
   **Reset** function, 5-25



- `~www_Server` destructor, 5-46
- `GetEchoFormsFlag` function, 5-47
- `GetEchoRequestsFlag` function, 5-46
- `Go` function, 5-46
- `SetEchoFormsFlag` function, 5-46, 5-47
- `SetFileNames` function, 5-47
- `SetPipeName` function, 5-47
- `www_Server` constructor, 5-46
- `www_Template` class
  - `~www_Template` destructor, 5-48
  - `GetForm` function, 5-49
  - `SetFormContents` function, 5-48
  - `www_Template` constructors, 5-48
- `www_User` class
  - `~www_User` destructor, 5-49
  - `AssignValue` function, 5-49
  - `ForEach` function, 5-50
  - `GenerateURL` function, 5-50
  - `GetMacroList` function, 5-49
  - `GetValue` function, 5-50
  - `IfCond` function, 5-50
  - `IncludeTemplate` function, 5-49
  - `www_User` constructor, 5-49
- `www_Wild` class
  - `~www_Wild` destructor, 5-51
  - `BuildSGMLToCURL` function, 5-51
  - `GetCitation` function, 5-51
  - `GetNotation` function, 5-51
  - `MakeLink` function, 5-51
  - `www_Wild` constructor, 5-51



