

## APPENDIX: COMPUTER CODE FOR THE SEARCH ENGINE

This appendix presents the verbatim computer-program code developed to create the search engine for Federal Standard 1037C, *Glossary of Telecommunication Terms*. Annotations and descriptions of important program components are included.

- ⇒ The following line indicates to the Perl language processor where to find the supporting Perl binary files.

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

- ⇒ The following lines set the initial variables for the file locations. They indicate where to find the internal database files and how to provide the correct base HTML link for creating hyperlinks.

```
# -- Setting base variables
$base = "http://www.its.blrdoc.gov/fs-1037";
$dir = "/nd/bing/public_html/1037e";
$ibase = "/nd/bing/public_html/pub";
$hbase = "http://www.its.blrdoc.gov/fs-1037";
```

- ⇒ The following lines define the basic weighting factors to determine the strength of a phrase match. See Section 2.1.2 for complete details of the weighting factors.

```
$weight1 = 10; # weighting factor if phrase in title
$weight2 = 30; # weighting factor if phrase IS title
$weight3 = 5; # weighting factor if phrase is whole word
$weight4 = 2; # weighting factor if phrase is in a definition
$weight5 = 10; # weighting factor if all phrases found
$maxhits = 4;
```

- ⇒ The following lines begin creating the HTML page that results from a search. The lines are the basic HTML codes that are required at the start of all HTML pages.

```
# -- Print out the basic header info
print "Content-type: text/html\n\n";
print "<BODY background=\"\$base/gifs/\$-backg2.gif\" text=\"\#000000\"
bgcolor=\"\#ffffff\" link=\"\#FF0000\" vlink=\"\#0000ff\" alink=\"\#00FF00\">\n";
print "<Style> A.HL0 {background-color: white; color: silver}</Style>\n";
print "<Style> A.HL1 {background-color: #ffffcc; color: blue}</Style>\n";
print "<Style> A.HL2 {background-color: lightgreen; color: green}</Style>\n";
print "<Style> A.HL3 {background-color: pink; color: red}</Style>\n";
print "<html><title>Search Results</title>\n";
print "<h1><center>Search Results</center></h1>\n";
```

- ⇒ The following lines start the internal timer. When the search is complete, the current time will be compared with this starting value to determine how long the search took. The next line

sets the number of matching terms to zero. As each new match is found, this value will be increased by one.

```
# -- Start the timer
$start = time;
$nhits = 0;
```

⇒ These lines cause the program to read in the command-line variables, which were created by the Web page that called this Perl script. The two following lines are “commented out” (by placing the “#” sign in front of them) and were used to print out each variable on the monitor, as it was read into the program, during testing and design of the program.

```
# -- Read in the variables from the command line from Web page
&ReadParse;
#foreach $key (keys %in) {
#  print "$key: $in{$key}<br>\n"; }
```

⇒ The following lines analyze the command-line variables and set defaults for those variables where no valid input was given by the user. If the user did not type a search phrase, the first set of lines will insert the word “space” as the search phrase. The second set will set the number of items to display to 25 if the user has not specified a number. The third set of lines will set the default document to search to be Federal Standard 1037C if none is specified. This option is commented out, since, at this time, Federal Standard 1037C is the only on-line Federal Standard that can be searched with this program.

```
# -- Print out warning messages and set default values
print "<ul>\n";
if ($in{searchphrase} eq "") {
  print "<li>You didn't type anything into the search blank, so I did a search
for '<i>space</i>' (note the irony).<br>\n";
  $in{searchphrase} = "space";
}

if ($in{just25} eq "") {
  print "<li>You didn't select how many results to display, so I limited the
display to the <i>top 25</i> matches.<br>\n";
  $in{just25} = "25"
}

# if ($in{tosearch} eq "") {
#  print "<li>You didn't select a document to search, so I searched <i>FS-
1037C</i>.<br>\n";
#  $in{tosearch} = "37"
# }
```

⇒ The next lines will break up a multi-word search phrase into separate words. They will also recognize that some search phrases are inside quotation marks and will treat those phrases as single words. These lines of code also try to “sanitize” the search words by eliminating any symbols that will create incorrect search results. The period is one such character that will create incorrect results, since Perl sees a period as a wild-card indicator (*i.e.*, a period means

“any one character”). Also, all lowercase letters are converted to uppercase letters. Later, during the actual search, the case of the letters is ignored.

```
# -- Break up searchphrase
$oldphrase = $in{searchphrase};
$in{searchphrase} =~ s/([^\a-zA-Z0-9_ \r\t\n\f"?\*])/\$1/g;
#print "$in{searchphrase}\n"; die;

$xsearch = $in{searchphrase};
while ($xsearch =~ /\"/) {
    $ysearch = $xsearch;
    $xsearch =~ s/(.*)\"(.*)\"(.*)/$1$3/;
    $ysearch =~ s/(.*)\"(.*)\"(.*)/$2/;
    #print "$ysearch\n";
    push (@wordlist,$ysearch);
}

$xsearch =~ s/,//g;
$xsearch =~ s/^\s*//g;
$xsearch =~ s/\s+//g;
$xsearch =~ s/\s*$//g;
@word2 = split (/ /,$xsearch);
foreach $i (@word2) {
    #print "$i\n";
    if ($i eq "and" || $i eq "or") {} else {
        push (@wordlist,$i)
    }
}
# push (@wordlist,@word2);
```

⇒ These lines are the start of the actual search algorithm. They call up each word in the search list and start the main search loop for each one.

```
foreach $i (@wordlist) {
    $wordhits{$i} = 0;
}
```

⇒ For each word, the database of terms is accessed from a file on the hard disk. Each line of the database is read in, one at a time.

```
# -- Open Database File and read each definition
open (FSDB, "$nbase/fsdb.txt");
while ($line = <FSDB>) {
    chop($line);
```

⇒ A line containing just “@@” indicates that the text of the current definition is complete and that the text of the next definition will begin with the next line in the file. If this is the case, the first line of the next definition is read in, and the term name is extracted from the line of text.

```
# -- For each file, set defaults
if ($line eq "@@") { # Indicates next definition
    # and begin special processing.
```

```
&push_stack;
$href = <FSDB>;
chop($href);
$fname = $href;
$fname =~ s/.*(dir-...\/_....\.htm).*/$1/;
$def = $href;
$def =~ s/.*main1">(.*)</a>/$1/;
```

- ⇒ The next lines instruct the computer to search for each of the target words that the user entered for each line that has been read into the Perl script. The script sets the initial “best match” variable to 0.25.

```
# -- Try each phrase
foreach $searchphrase (@wordlist) {
    $hit{$searchphrase} = 0.25;
```

- ⇒ If the target word is found in the first line of the text of the definition, the “best match” variable is increased by the weighting variable (weight1) that represents a simple match. If the target word starts the line of text (*i.e.*, the target word is the term name), the appropriate weighting factor (weight2) is added to the “best match” variable. Similarly, if the target word is found as a whole string, separated by spaces, the appropriate weighting variable (weight3) is added to the “best match” variable.

```
# -- Add extra if hit is in Def.
if ($def =~ /$searchphrase/i ) {
    $besthit = $besthit + $weight1;
    if ($def =~ /^$searchphrase$/i ) { $besthit = $besthit + $weight2 }
    if ($def =~ /\W$searchphrase\W/i ) { $besthit = $besthit + $weight3 }
    if ($thisword{$searchphrase} == 0) {
        ++$thisword{$searchphrase};
        ++$hit{$searchphrase}
    }
} #End if in Def name
} #End each phrase
```

- ⇒ The target word is sought within each additional line in the text of the definition. If it is found, a percentage of the final weighting factor (weight4) is added. This percentage is determined by calculating how close to the beginning of the line of text the match occurs. Only the first four matches within the text of the definition are considered.

```
} else {
    # If not new def, do normal processing

    # -- Try each phrase
    foreach $searchphrase (@wordlist) {

# -- and then examine each line
$before = length($line); # Determine line length
#print "$line\n";
$liney = $line;
while ($liney =~ /$searchphrase/i) {
    if ($thisword{$searchphrase} == 0) {
```

```

++$thisword{$searchphrase};
++$hit{$searchphrase};
}
if ($liney =~ /\W$searchphrase\W/i) {
$liney =~ s/(.*)$searchphrase.*/$1/i;
$lafter = length($liney); # Determine chopped line length
$besthit = $besthit + $weight3 * (1 - ($lafter/$lbefore));
} else {
$liney =~ s/(.*)$searchphrase.*/$1/i;
$lafter = length($liney); # Determine chopped line length
$besthit = $besthit + $weight4 * (1 - ($lafter/$lbefore));
}
} # End 'found phrase' While loop
} # End for each phrase

```

- ⇒ If the user has entered multiple words or phrases to search for, and if several of them are found within one definition, an extra weight is added (weight5) to the “best match” variable. This extra weight is multiplied by a factor determined by how often each word is found (to the limit of four).

```

## -- Calculate weight5
$j = $weight5 * $#wordlist; $j2 = $j;
#print "$j\n"; die;
foreach $i (@wordlist) {
if ($hit{$i} > $maxsize) {$hit{$i} = $maxsize}
$j = $j * $hit{$i}; $j2 = $j2 *.25;
}
if ($j != $j2) {$besthit = $besthit + $j}

```

- ⇒ These next lines are the end of the search loop. The loop is executed once for each definition in the database.

```

} # End if @@
} # end while
&push_stack;

#$nword = $in{word};
#print "Best Score: $verybest<br>\n";

```

- ⇒ These lines then analyze the scores accumulated by each definition. The definitions that score the highest weighted values will rise to the top of the list. The first few lines calculate the amount of time it took to search the database.

```

# -- Sort and print the files by most likely hit
$stend = time;
$stsecs = $stend - $ststart;
print "<li><font color = red>5860</font> definitions searched in <font color =
blue>$stsecs</font> seconds (actual time to transmit results to your browser
may have taken longer)\n";

```

- ⇒ These lines calculate and display the number of definitions that contained any of the matching phrases.

```
print "<li><font color = red>$nhits</font> definition";
  if ($nhits != 1) {print "s"}
print " found that contain the item";
  if ($#wordlist > 0) {print "s"}
print " ";
```

- ⇒ These lines separate the hits by the different target search words. The number of hits for each word is displayed on the Web page.

```
$i = -1;
foreach $word (@wordlist) {
  $wh = $wordhits{$word};
  $word =~ s/\\([^\s])/g;
  print "\"<font color = green><b>$word</b></font>\"";
  if ($#wordlist > 0) { print " (<font color = red>$wh</font>)" }
  ++$i;
  if ($i == $#wordlist) {
    print ".\n</ul><hr>\n";
  } else {
    if ($i == ($#wordlist - 1)) {print " and/or " } else {print ", " }
  }
}
```

- ⇒ If the search returns no matches at all, these next few lines display a simple list of possible reasons why no matching items were found. The reasons range from simple typographical errors to the absence of the typed words anywhere within the text of the definitions.

```
if ($nhits == 0) {print "<ul><li>Here are some possible <a href=\""$hbase/fs-0.htm\" target=\"main1\">reasons</a> why you found no matches.</ul>\n"} }
```

- ⇒ These lines sort the matches to present the highest weighted matches first, followed by matches with lower weights.

```
$i = 0;
foreach $fname (sort { $hitsize{$b} <=> $hitsize{$a} } keys %hitsize) {
  #print "$fname, $hitsize{$fname}, $hitname{$fname}\n";
  # $perc = int(10000*$hitsize{$fname}/$verybest)/100;
  ++$i; if ($i > $in{just25}) {last}
  $perc = int(100*$hitsize{$fname}/$verybest+.5);
```

- ⇒ These lines will provide a color background appropriate to the weighting of the individual term name listed in the results. Those matching definitions that score above 66.6% will be displayed in pink, those definitions that score between 33.3% and 66.6% will be shown in light green and those definitions that score below 33.3% will be shown in yellow.

```
$highl = 0;
  if ($perc >= 0) { $highl = 1 }
  if ($perc > 33.33) { $highl = 2 }
  if ($perc > 66.67) { $highl = 3 }

  print "<i>$perc%</i> -- <a class=HL$highl $hitname{$fname}</a></style><br>\n";
}
```

- ⇒ These lines provide the “wrap up” text for the HTML Web page. They provide a blank for the user to begin another search and then they provide the final closing lines that are required in all Web pages.

```
print "<hr>Try Another Quick Search!<br>\n";
print "<FORM METHOD=\"POST\" ACTION=\"http://132.163.64.201/cgi-
bin/searchfs.prl\" TARGET=\"main2\">\n";
print "Phrase to Search for:<br><INPUT NAME=\"searchphrase\" SIZE=25><br>\n";
print "<br>How many matches to show:";
print "<br><INPUT TYPE=\"radio\" NAME=\"just25\" VALUE=\"25\">Top 25";
print "<INPUT TYPE=\"radio\" NAME=\"just25\" VALUE=\"100\">100";
print "<INPUT TYPE=\"radio\" NAME=\"just25\" VALUE=\"6000\">All";
print "</form>";
print "</html>\n";
```

- ⇒ This subroutine, written by ITS staff, adds the name of a definition (if any match is found) to the results “stack” of definitions that include matches. All of the associated data (*i.e.*, the “best match” value) is pushed onto a parallel stack.

```
# -----
# -- Push hits onto stack subroutine
sub push_stack {
  foreach $i (@wordlist) {
    if ($thisword{$i} > 0) {++$wordhits{$i};
    #print "<br>WHIT($wordhits{$i}) -
$liney<br>\n$besthit<br>\n$wordlist{$i}<br>\n";
    $thisword{$i} = 0; }
  }
  if ($besthit > 0 ) {
    #print "$liney\n$def\n$fname\n$besthit\n$wordlist{$searchphrase}\n";
die;
    if ($besthit > $verybest) { $verybest = $besthit }

    # print "$fname, $i -- ";
    ++$nhits; # Count the number of hit files
    #print "<br>NHIT($nhits) -
$liney<br>\n$besthit<br>\n$wordlist{$i}<br>\n";
    $hitsize{$fname} = $besthit;
    $hitname{$fname} = $href;
  } # End if
  $besthit = 0;
} # End stack push
```

- ⇒ The following subroutine<sup>4</sup> reads command-line variables generated by the search-engine Web page and based on the text that the user types into the blanks on the Web page. In this instance, the variables are the target search phrase and the number of matching entries to return.

<sup>4</sup> Permission to use this subroutine was granted by the owners of the copyright, via the Web page where this, and other useful Perl subroutines, are made available for Perl programmers. That page is, “*The cgi-lib.pl Home Page*,” and is located at URL, <http://cgi-lib.stanford.edu/cgi-lib/>. This Web page was last accessed on March 5, 1999.

```
# -----
# Adapted from cgi-lib.pl by S.E.Brenner@bioc.cam.ac.uk
# Copyright 1994 Steven E. Brenner
sub ReadParse {
    local (*in) = @_ if @_;
    local ($i, $key, $val);

    if ( $ENV{'REQUEST_METHOD'} eq "GET" ) {
        $in = $ENV{'QUERY_STRING'};
    } elsif ( $ENV{'REQUEST_METHOD'} eq "POST" ) {
        read(STDIN,$in,$ENV{'CONTENT_LENGTH'});
    } else {
        # Added for command line debugging
        # Supply name/value form data as a command line argument
        # Format: name1=value1\&name2=value2\&...
        # (need to escape & for shell)
        # Find the first argument that's not a switch (-)
        $in = ( grep( !/^-/, @ARGV ) ) [0];
        $in =~ s/\\&/&/g;
    }

    @in = split(/&/,$in);

    foreach $i (0 .. $#in) {
        # Convert plus's to spaces
        $in[$i] =~ s/\+/ /g;

        # Split into key and value.
        ($key, $val) = split(=/,$in[$i],2); # splits on the first =.

        # Convert %XX from hex numbers to alphanumeric
        $key =~ s/%(..)/pack("c",hex($1))/ge;
        $val =~ s/%(..)/pack("c",hex($1))/ge;

        # Associate key and value. \0 is the multiple separator
        $in{$key} .= "\0" if (defined($in{$key}));
        $in{$key} .= $val;
    }
    return length($in);
}
```

This program was written by ITS staff under the direction and sponsorship of NCS.