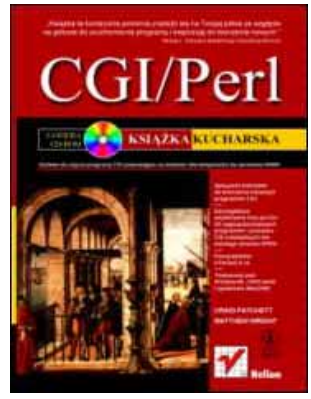
CGI / Perl - książka kucharska



Autorzy: Craig Patchett, Matthew Wright
Tłumaczenie: Michał Mosiewicz, Jerzy Hodor
ISBN: 83-7197-061-7
Format: B5, 528 stron
Zawiera CD-ROM
Data wydania: 07/1999
Cena książki: 65.30 zł
Przesyłka gratis! Odbiorca pokrywa jedynie koszty pobrania (2,70 zł) w przypadku przesyłki za zaliczeniem pocztowym

Wydawnictwo Helion
ul. Chopina 6, 44-100 Gliwice, POLAND
telefon: (32) 230-98-63, 231-22-19
fax: (32) 230-98-63 w.10
mail: helion@helion.com.pl

Książka "CGI/Perl" zaczyna się od zwięzłego opisu zasad tworzenia programów CGI dla początkujących oraz przewodnika prezentującego sposoby instalacji tych programów na serwerach WWW działających w systemach UNIX, Windows oraz MacOS. Pozostała część książki zawiera szczegółową analizę siedmiu doskonałych programów CGI, wśród których będziesz mógł znaleźć: mechanizm wyszukiwawczy, system kontroli dostępu, zaawansowane przetwarzanie formularzy, karty zakupowe, obsługę poczty elektronicznej, kodowanie plików, weryfikację kart kredytowych, sprawdzanie poprawności adresów poczty elektronicznej, obsługę błędów oraz szyfrowanie tekstu. Wraz z każdym programem podane zostały dodatkowe informacje dotyczące sposobów jego instalacji oraz obsługi, jak również profesjonalne rady i podpowiedzi dotyczące tworzenia programów CGI w języku Perl .

Zobacz przykładowy rozdział

Spis treści

Jeżeli znasz tę książkę oceń ją

Aktualny cennik książek e-mailem

Książki i "3D" Online

Informacje o nowościach e-mailem

Zamów najnowszy katalog

Rozdział 5.

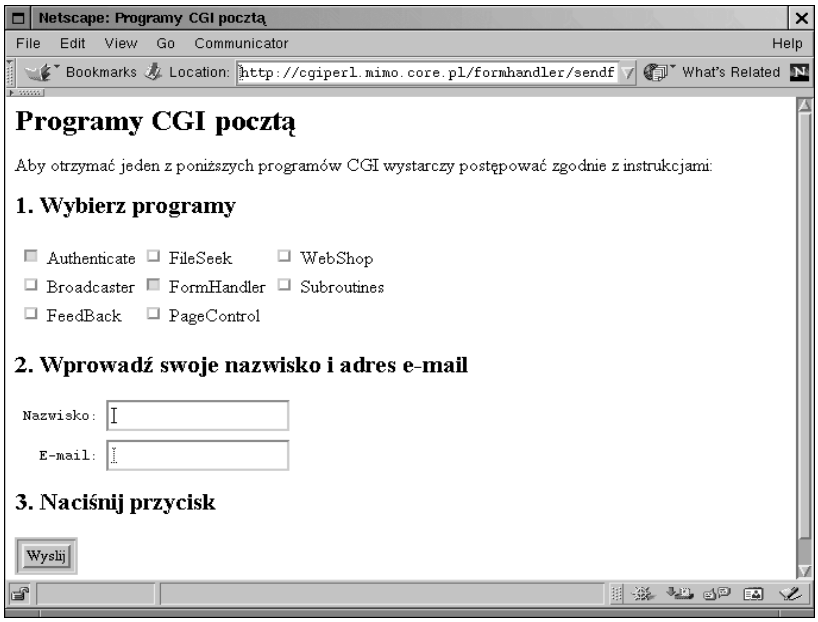
Pobieranie danych: formhandler.cgi

Kluczem do tworzenia interaktywnych serwisów WWW jest stosowanie formularzy HTML-owych, które pozwalają na pobieranie danych od odwiedzających. Niestety, mimo łatwości zbierania danych, nie ma domyślnych mechanizmów ich przetwarzania. Bez programu CGI nie można zobaczyć, czy zostały wypełnione wszystkie wymagane pola, nie można również zapisać danych w bazie danych, nie można ich przesłać pocztą (w czytelnym formacie), nie można nawet potwierdzić ich odbioru. W rozdziale tym przedstawimy program, który zajmuje się wszystkimi podstawowymi czynnościami, oferując jednocześnie bardziej zaawansowane możliwości, dzięki którym można z łatwością użyć formularzy do przesyłania plików pocztą, tworzenia interfejsów do poczty przez WWW, administrowania prostymi listami dyskusyjnymi (wysyłania grupowych wiadomości) i do innych zadań.

Możliwości

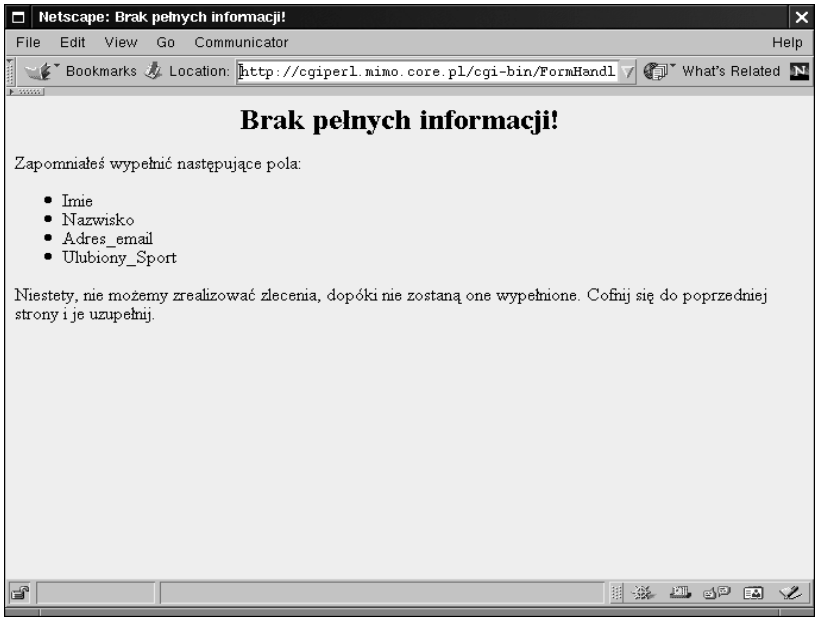
Istnieje wiele gotowych programów CGI przeznaczonych do przetwarzania danych z formularzy. Dlatego też przy projektowaniu programu *FormHandler* staraliśmy się uwzględnilić wiele możliwości, których nie znajdziemy w innych programach. Tak więc, program będzie obsługiwał nasze podstawowe potrzeby w zakresie formularzy, dając równocześnie możliwość budowy sporych interaktywnych serwisów, co pokażemy w dalszych przykładach. Zaczniemy od ogólnego opisu możliwości, który da nam pogląd na działanie programu *FormHandler*.

Całkowita konfigurowalność. Zanim przejdziemy do omówienia innych właściwości programu, powinniśmy zwrócić uwagę na to, że został on zaprojektowany w sposób umożliwiający jego swobodną konfigurację, co oznacza, że można go skonfigurować tak, by obsługiwał tylko te opcje, na których nam zależy. Mamy również pełną kontrolę nad wyglądem, układem i zawartością informacji prezentowanej odwiedzającym.



Ilustracja 5.1. *Przykładowy formularz programu FormHandler obsługujący wysyłanie plików pocztą elektroniczną*

Możliwość oznaczenia pól wymaganych. *FormHandler* pozwala na określenie pól, których wypełnienie jest wymagane i wyświetli wiadomość z listą pól pominiętych przez odwiedzającego przy wypełnianiu formularza (zobacz ilustracja 5.2). Dzięki temu mamy pewność, że otrzymamy wszystkie niezbędne informacje.



Ilustracja 5.2. *Przykład komunikatu o niewypełnieniu wszystkich wymaganych pól*

Ilustracja 5.3. *Przykład strony potwierdzającej wypełnienie formularza*

Opcjonalna strona potwierdzenia. Kiedy formularz zostanie poprawnie wypełniony i wysłany, można wyświetlić potwierdzenie podając adres URL istniejącej strony bądź za pomocą własnego pliku szkieletu strony. Własny szkielet strony pozwala na wyświetlenie wartości wprowadzonych pól. Przykład takiej strony przedstawiony jest na ilustracji 5.3.

Opcjonalne powiadomienie administratora. Po wypełnieniu formularza program może automatycznie przesłać list elektroniczny pod jeden lub więcej adresów z podsumowaniem zawartości formularza. Dzięki temu administrator może być automatycznie powiadamiany o każdym wypełnionym formularzu.

Opcjonalne powiadomienie odwiedzającego. Program oferuje również możliwość wysyłania do odwiedzającego własnego powiadomienia pocztą elektroniczną na adres podany w wypełnionym formularzu. Możemy w sposób dowolny konfigurować format wiadomości. Może zawierać ona proste podziękowanie lub podsumowanie danych zawartych w formularzu. Wartości poszczególnych pól mogą również zostać dołączone do treści listu, umożliwiając, na przykład, odwołanie się w treści listu do nazwiska odwiedzającego.

Dołączanie plików do powiadomienia. *FormHandlera* można również użyć do wysyłania dodatkowych informacji lub plików drogą pocztową. Zostaną one dołączone do listu za pomocą standardowych protokołów pocztowych. (Ilustracja 5.4 zawiera przykład listu zawierającego dołączony plik).

Opcjonalny plik logu. Istnieje również możliwość takiego skonfigurowania programu, by zamiast lub w uzupełnieniu do wysyłania nam wypełnionych formularzy pocztą elektroniczną zapisywał ich zawartość do odpowiedniego pliku. Jego zawartość i format można określić za pomocą pliku wzorca lub wyszczególniając listę pól i separator używany do ich rozdzielenia. Program pozwala również na dodawanie indywidualnych identyfikatorów do każdego z rekordów.

Ilustracja 5.4. *Przykład wiadomości e-mailowej wysłanej po wypełnieniu formularza*

```
From: jas@domena.com
To: sportowy@sklep.com
Subject: Cennik wyslany

Cennik zostal wyslany (6/05/1997 23:13)

Nazwisko: Jan Kowalski
Email: jas@domena.com
Wyslano plik: /home/cennik/cennik1.txt
Browser: Mozilla/4.0 (Macintosh; I; PPC)
```

Opcjonalna lista dystrybucyjna. *FormHandler* zawiera również unikalną możliwość tworzenia i zarządzania listą dyskusyjną w oparciu o budowany automatycznie spis adresów poczty elektronicznej oraz plik dziennika pracy (log). Dzięki temu odwiedzający mogą sami dopisywać się do listy subskrybentów oraz wysyłać na nią swoje wiadomości. Całość działa automatycznie poprzez interfejs WWW. Przykład wykonania takiej listy znajduje się w dalszej części tego rozdziału.

Możliwość określenia porządku pól. Kolejność pól formularza, które są przesyłane za pomocą protokołu CGI, nie daje się z góry określić, dlatego program umożliwia określenie ich porządku podczas generowania listy pól nie wypełnionych, stron podsumowań oraz wiadomości pocztowych.

Możliwość używania opisowych nazw pól. HTML narzuca ograniczenia co do sposobu nazywania pól. *FormHandler* usuwa te ograniczenia. Za każdym razem, gdy zachodzi potrzeba wyświetlenia nazwy jakiegoś pola (na przykład na stronie o brakujących danych), może zostać mu przypisana bardziej opisowa, dłuższa nazwa.

Obsługa wielu formularzy. Program został zaprojektowany w sposób pozwalający na jednoczesną obsługę wielu formularzy za pomocą pojedynczej kopii programu. (Jest to szczególnie użyteczne, jeśli nie mamy łatwego dostępu do katalogu cgi-bin serwerze).

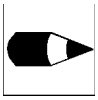
Opcje bezpieczeństwa. Program zawiera kilka opcji bezpieczeństwa, które pozwalają na nałożenie limitów na to, które pliki mogą być przesyłane pocztą i kto może dołączać formularze do programu.

Przewodnik użytkownika

Korzystanie z programu *FormHandler* jest nadzwyczaj proste i składa się z dwóch czynności. Po pierwsze – należy zainstalować i skonfigurować sam program. Po drugie – należy przygotować formularz oraz pliki wzorców. Następnie wystarczy wypełnić formularz i go wysłać. Instalacja i konfiguracja programu jest omówiona w kolejnych podrozdziałach. Powiemy również, jakie pliki trzeba przygotować i jak tego dokonać.

Użycie zmiennych konfiguracyjnych formularza

Zmienne konfiguracyjne są zdefiniowane w tablicy asocjacyjnej %CONFIG programu (zob. komentarze w liniach 49. do 62.). Mimo że można zdefiniować ich domyślne wartości wewnątrz samego programu (na przykład określić adres e-mailowy administratora), w większości przypadków będziemy umieszczać w formularzach pola o nazwach odpowiadających zmiennym, jakie chcemy zdefiniować. Skorzystamy w tym celu z pól ukrytych, aby zdefiniować wartości stałe, lub z pól wypełnianych przez użytkownika – dla określenia wartości zmiennych. Zatem, jak widać, możliwe jest takie skonfigurowanie programu, by był on zależny tylko od treści formularza.



Aby program poprawnie działał, należy zdefiniować zmienne konfiguracyjne dokładnie tak jak poniżej – przy pomocy tablicy asocjacyjnej **%CONFIG** lub za pomocą pól formularzy, stosując odpowiednio małe litery i podkreślenia.

Zmienne kontrolne formularzy

Pierwsza grupa zmiennych konfiguracyjnych formularzy, której się przyjrzymy, kontroluje sposób przetwarzania i wyświetlania pól.

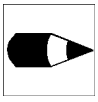
required jest opcjonalną zmienną, zawierającą listę pól oddzielonych przecinkami, których wartości muszą zostać bezwzględnie podane przez użytkownika (np. '`realname,email`'). Ukryte pole formularza może mieć następującą postać:

```
<INPUT TYPE="hidden" NAME="required" VALUE="realname,email">
```

sort jest opcjonalną zmienną, która może zostać zdefiniowana w celu określenia kolejności, w jakiej lista pól zostanie przesłana pocztą lub przedstawiona na stronach HTML-owych. Jeśli pola mają zostać wysłane w układzie alfabetycznym, wystarczy ustawić jej wartość na '`alphabetic`'. Aby posortować pola w określonym porządku, należy użyć konstrukcji 'order:', po której następuje lista pól w porządku, w jakim mają się one pojawiać (np. '`order:addr,city,state,zip`'). Przykładowe, ukryte pole formularza może wyglądać następująco:

```
<input type="hidden" NAME="sort" VALUE="order:addr,city,state,zip">
```

Jeśli nie ustawimy tej zmiennej, pola będą wyświetlane w porządku przypadkowym. Jeśli określimy porządek, ale nie podamy przy tym wszystkich pól, wtedy te, które zostały pominięte, zostaną wyświetlone na końcu w porządku przypadkowym.



FormHandler będzie wyświetlał wszystkie zmienne konfiguracyjne określone w **print_config** na początku, bez względu na określoną kolejność sortowania.

field_names jest opcjonalną zmienną, która może zawierać listę par wartości zawierających bardziej opisowe nazwy zmiennych. Każde pole i jego opis powinny być rozdzielone znakiem równości, a poszczególne pary znakiem ampersand (&):

```
realname=Twoje nazwisko&address=Twój adres&Zip=Kod pocztowy
```

Ukryte pole formularza może wyglądać następująco:

```
<INPUT TYPE="hidden" NAME="field_names" VALUE="realname=Twoje
  nazwisko&address=Twój adres&Zip=Kod pocztowy">
```

print_config jest zmienną opcjonalną, która może zawierać listę zmiennych konfiguracyjnych, które program ma umieszczać w wiadomościach pocztowych i generowanych stronach HTML-owych. Domyślnie żadna z tych zmiennych nie jest wyświetlana. Przykład ukrytego pola konfigurującego wartość zmiennej może być następujący:

```
<INPUT TYPE="hidden" NAME="print_config" VALUE="realname,email">
```

env_report jest opcjonalną zmienną, której może być przypisana lista zmiennych środowiskowych CGI. Lista tych zmiennych wraz z odpowiadającymi im wartościami będzie dołączana do każdego listu elektronicznego wysłanego na adres odbiorcy po wypełnieniu formularza (np. '`HTTP_USER_AGENT, REMOTE_HOST`'). Ukryte pole formularza może mieć postać następującą:

```
<INPUT TYPE="hidden" NAME="env_report" VALUE="HTTP_USER_AGENT,REMOTE_HOST">
```

Lista wszystkich zmiennych środowiskowych CGI znajduje się w dodatku B.

Zmienne identyfikujące odwiedzającego

Następna grupa zmiennych służy do przechowywania różnych informacji o odwiedzającym.

realname jest zmienną opcjonalną, która może zawierać pełne imię i nazwisko odwiedzającego, wpisywane przy pomocy pola tekstowego (np. 'Jan Kowalski'):

```
<INPUT TYPE='text' name="realname">
```

Jeśli chcesz, można zamiast pola `realname` użyć kombinacji dwóch pól określających imię i nazwisko (następne dwie zmienne).

first_name jest opcjonalną zmienną, która zawiera imię odwiedzającego, określone za pomocą pola tekstowego (np. 'Jan'):

```
<INPUT TYPE='text' NAME="first_name">
```

last_name jest opcjonalną zmienną, która zawiera nazwisko odwiedzającego, określone za pomocą pola tekstowego (np. 'Kowalski').

```
<INPUT TYPE='text' NAME="last_name">
```

email jest zmienną nieopcjonalną (w większości przypadków), która określa adres poczty elektronicznej odwiedzającego (typu 'odwiedzający@domena.com'). Można ją określać za pomocą pola tekstowego:

```
<INPUT TYPE='text' NAME="email">
```

Zmienne pocztowe

Trzecia grupa zmiennych konfiguracyjnych określa sposób przetwarzania poczty. Pamiętajmy, że *FormHandler* jest zaprojektowany do obsługi dwóch typów wiadomości pocztowych. (Oba są opcjonalne). Pierwszy typ to informacje przekazywane do administratora serwisu po prawidłowym wypełnieniu formularza (jak zobaczymy w dalszej części rozdziału, można z nich również skorzystać w inny sposób). Wiadomości te będziemy nazywać powiadomieniami. Drugi typ to wiadomości wysyłane do odwiedzającego, które będziemy nazywać odpowiedziami.

subject jest zmienną opcjonalną, która określa temat powiadomienia (np. 'Dane z ankiety'). Ukryte pole formularza określające ją powinno mieć następującą postać:

```
<INPUT TYPE="hidden" NAME="subject" VALUE="Dane z ankiety">
```

recipient jest wymaganym polem, które określa odbiorcę powiadomienia (np. '`użytkownik@domena.com`'). Najczęściej będzie to nasz własny adres e-mailowy. Można go ustawić stosując ukryte pole:

```
<INPUT TYPE="hidden" NAME="recipient" VALUE="uzytkownik@domena.com">
```

Jeśli będziemy przechowywać logi formularzy i nie chcemy otrzymywać za każdym razem pocztowego powiadomienia o wypełnieniu formularza, możemy ustawić jej wartość na 'none' lub pozostawić ją niezdefiniowaną.

email_template jest opcjonalną zmienną określającą pełną ścieżkę do pliku zawierającego wzór listu powiadomienia (np. '/home/web/formhandler/email.txt'). Odpowiednie pole formularza może mieć następującą postać:

```
<INPUT TYPE="hidden" NAME="email_template"
 VALUE=/home/web/formhandler/email.txt">
```

Jeśli zmienna ta nie jest zdefiniowana, ale zdefiniowano adres odbiorcy powiadomienia, wtedy *FormHandler* wyśle list o predefiniowanej postaci.

cc jest zmienną opcjonalną, określającą dokąd powinna zostać wysłana kopia powiadomienia. Może zawierać listę adresów odbiorców oddzielonych przecinkami (np. '`użytkownik1@domena.com,użytkownik2@domena.com`') lub słowo kluczowe `file`, po którym wypisana jest po dwukropku ścieżka do pliku przechowującego w kolejnych liniach listę adresów (np. '`file:/home/web/formhandler/cc.txt`'). Ukryte pole ustawiające tę wartość może mieć jedną z poniższych postaci:

```
<INPUT TYPE="hidden" NAME="cc"
 VALUE="uzytkownik1@domena.com,uzytkownik2@domena.com">

<INPUT TYPE="hidden" NAME="cc" VALUE="file:/home/web/formhandler/cc.txt">
```

bcc jest zmienną opcjonalną, która określa, kto powinien otrzymać tzw. „blind copy” listu potwierdzenia. Odbiorcy takiego listu nie są jawnie określeni w jego nagłówkach. Wartość tej zmiennej ustawiamy w podobny sposób jak poprzedniej.

mail_list jest zmienną opcjonalną, która określa listę odbiorców potwierdzenia. Ma taką samą postać jak poprzednio omówione zmienne cc i bcc.

reply_message_subject jest opcjonalną zmienną, służącą do określenia tematu odpowiedzi wysyłanej do odwiedzającego (np. 'Witaj w naszym serwisie!'). Ukryte pole formularza ustawiające wartość tej zmiennej może wyglądać następująco:

```
<INPUT TYPE="text" NAME="reply_message_form" VALUE="Witaj w naszym
 serwisie!">
```

Jeśli nie ustawimy tej zmiennej, program użyje nagłówka 'FormHandler Reply Message'.

reply_message_from jest opcjonalną zmienną, która określa adres zwrotny odpowiedzi wysyłanej do odwiedzającego (np. 'user@domena.com'). W większości przypadków będzie to nasz własny adres, taki sam jak w przypadku odbiorcy potwierdzenia. Zmienną tę można ustawić stosując ukryte pole formularza:

```
<INPUT TYPE="hidden" NAME="reply_message_from"
  VALUE="uzytkownik@domena.com">
```

Jeśli nie określimy wartości tej zmiennej, program skorzysta z zawartości zmiennej recipient.

reply_message_template jest opcjonalną zmienną, której może zostać przypisana ścieżka do pliku z wzorcem odpowiedzi wysyłanej do odwiedzającego (np. '/home/web/formhandler/reply.txt'). Ukryte pole formularza ustawiające wartość zmiennej może wyglądać następująco:

```
<INPUT TYPE="hidden" NAME="reply_message_template"
  VALUE="/home/web/formhandler/reply.txt">
```

Jeśli ani ta zmienna, ani zmienna reply_message_from nie będą ustawione lub brakować będzie adresu e-mailowego (email), wtedy odpowiedź nie zostanie wysłana.

reply_message_attach jest opcjonalną zmienną, która może zawierać listę plików rozdzielonych przecinkami, które *FormHandler* dołączy do listu. Każdy plik powinien być opisany w następującym formacie:

```
typ_kodowania:ścieżka_do_pliku
```

typ_kodowania jest kluczem (`text`, `uuencode` lub `mime`) określającym sposób kodowania pliku w liście, zaś ścieżka określa dokładnie jego położenie. Dołączane pliki muszą znajdować się w jednym z katalogów znajdujących się w tablicy @ALLOWED_ATTACH_ DIRS i mogą być wybierane przez użytkownika za pomocą pól wyboru (*checkbox*), listy wyboru lub można je podać jako listę ukrytych pól formularza. Lista wyboru może mieć przykładową postać:

```
<SELECT NAME="reply_message_attach">
  <OPTION VALUE="text:/home/formhandler/files/info.txt"> Wyślij Info (Tekst)
  <OPTION VALUE="mime:/home/formhandler/files/info.pdf"> Wyślij Info (PDF)
</SELECT>
```

Ponieważ *FormHandler* obsługuje wielokrotne wartości zmiennych, można zezwolić odwiedzającemu na wybór kilku plików w tym samym czasie, stosując listy wyboru z atrybutem MULTIPLE albo checkboxy o tej samej wartości pola *name*.

Zmienne konfiguracji logu

Kolejna grupa zmiennych konfiguracyjnych określa sposób zapisu danych w logu.

log_filename jest opcjonalną zmienną, która wskazuje ścieżkę do pliku w systemie zawierającego plik logu (np. '`/home/web/formhandler/log.txt`'). Można ją ustawić stosując ukryte pole formularza:

```
<INPUT TYPE="hidden" NAME="log_filename"
  VALUE="/home/web/formhadnler/log.txt">
```

Jeśli jej nie ustawimy, program nie będzie generował pliku logu.

log_template jest opcjonalną zmienną wskazującą plik zawierający wzór rekordu logu (np. '`/home/web/formhandler/log_template.txt`'). Można ją ustawić stosując ukryte pole formularza:

```
<INPUT TYPE="hidden" NAME="log_template"
  VALUE="/home/web/formhadnler/log_template.txt">
```

log_fields jest opcjonalną zmienną, która może zawierać listę pól, których wartości zostaną zapisane w logu (np. '`realname,email`'). Pola są zapisywane w kolejności wystąpienia i rozdzielone separatorem określonym przez zmienną log_delimiter.

```
<INPUT TYPE="hidden" NAME="log_fields" VALUE="realname,email">
```

Zmienna musi być stosowana w połączeniu z log_filename.

log_delimiter jest opcjonalną zmienną (wymaganą, jeśli korzystamy z logu bez wzorca rekordu), która może zawierać ciąg znaków służący do rozdzielania pól w rekordach (na przykład '`||`'). Zmienna może być ustawiona za pomocą następującej konstrukcji:

```
<INPUT TYPE="hidden" NAME="log_delimiter" VALUE="||">
```

Jeśli nie ustawimy wartości tej zmiennej, program przyjmie jako domyślną wartość tej zmiennej znak tabulacji.

log_uid jest opcjonalnym znacznikiem, określającym czy program ma automatycznie generować identyfikatory użytkowników. Jeśli zmienna zawiera cokolwiek innego niż pusty ciąg znaków, wtedy FormHandler będzie generował unikalny identyfikator dla każdego z wpisów w logu. Może on być użyty jako identyfikator do części serwera chronionej hasłem lub do budowy baz danych. Można go ustawić za pomocą następującej składni:

```
<INPUT TYPE="hidden" NAME="log_uid" VALUE="yes">
```

Inne zmienne

Ostatnia grupa zmiennych konfiguracyjnych określa, co odwiedzający zobaczy po poprawnym wypełnieniu formularza lub w przypadku błędu.

success_html_template jest zmienną opcjonalną wskazującą na wzór strony generowanej po prawidłowym wypełnieniu formularza (np. '`/home/web/formhandler/success.html`'). Można ją ustawić następująco:

```
<INPUT TYPE="hidden" NAME="success_html_template"
  VALUE="/home/web/formhandler/success.html">
```

redirect jest opcjonalną zmienną, która może zawierać URL do strony, do której odwiedzający zostanie odesłany po prawidłowym wypełnieniu formularza (np. '`http://www.domena.com/yes.html`'). Można ustawić jej wartość stosując ukryte pole formularza:

```
<INPUT TYPE="hidden" NAME="redirect"
  VALUE="http://www.domena.com/yes.html">
```

error_html_template jest opcjonalną zmienną, która może zawierać ścieżkę do wzorca odpowiedzi, generowaną w przypadku, gdy nie są wypełnione wszystkie wymagane pola. Można ją ustawić za pomocą ukrytego pola formularza:

```
<INPUT TYPE="hidden" NAME="error_html_template"
  VALUE="/home/web/formhandler/missing.html">
```

Jeśli zmienna nie jest ustawiona, *FormHandler* wyświetli predefiniowaną stronę.

Tworzenie formularza

Po przeanalizowaniu wszystkich zawiłych detali dochodzimy do wniosku, że tworzenie formularzy do wykorzystania z programem *FormHandler* polega po prostu na dodaniu do typowego formularza dodatkowych odpowiednich pól ukrytych oraz ustawienia atrybutu ACTION znacznika FORM tak, aby wskazywał na *FormHandler*. (Atrybut METHOD może mieć wartość zarówno POST, jak i GET). Zademonstrujemy ten proces korzystając z prostego formularza pokazanego na ilustracji 5.5. (Może nie jest on piękny, ale zawiera tylko minimalny zestaw znaczników HTML).

Pamiętajmy, że jedyną niezbędną zmienną formularza jest zmienna recipient (odbiorca potwierdzenia). Inne zmienne należy ustawić jedynie wtedy, gdy potrzebne nam są dodatkowe funkcje programu. (Oznacza to, że jeśli zmienna recipient została skonfigurowana wewnątrz programu, to nie musimy określać żadnych zmiennych i możemy stosować go w istniejących formularzach, pod warunkiem że ich atrybut ACTION wskazuje na lokalizację programu *FormHandler*). Na przykład ilustracja 5.6 zawiera kod HTML-owej strony pokazanej na ilustracji 5.5. Formularz, po wysłaniu, wysyła potwierdzenie do właściciela (jego przykład możesz zobaczyć na ilustracji 5.7) i wyświetla odwiedzającemu domyślną stronę potwierdzenia (zob. ilustracja 5.8).

Ilustracja 5.5. *Prosty formularz, który wykorzystujemy w przykładach*

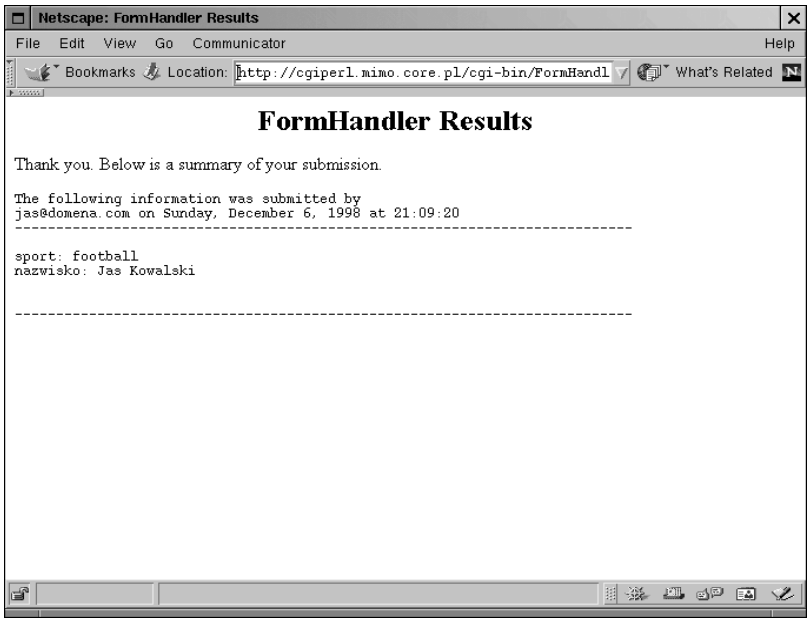
Ilustracja 5.6. *Kod HTML-owy prostego formularza do wykorzystania z FormHandlerem*

```
<HTML>
   <HEAD>
      <TITLE>Sports Stuff: Price List Request</TITLE>
   </HEAD>
   <BODY BGCOLOR="EEEEEE">
      <H1>Price List Request</H1>
      <P>
      <B>Enter your name and email address, and select the
      sport you're most interested in, and we'll send out the
      price list right away.</B>
      <P>
      <FORM ACTION="/cgi-bin/FormHandler.cgi" METHOD="POST">
      First Name: <INPUT TYPE="TEXT" NAME="first_name"
        MAXLEN="50" SIZE="30"><BR>
      Last Name: <INPUT TYPE="TEXT" NAME="last_name"
         MAXLEN="50" SIZE="30"><BR>
      <P>
      Email: <INPUT TYPE="TEXT" NAME="email" MAXLEN="50"
        SIZE="30"><BR>
      <P>
      Sport: <SELECT NAME="reply_message_attach">
             <OPTION VALUE="">--Select Here--
             <OPTION VALUE="baseball">Baseball
             <OPTION VALUE="football">Football
             <OPTION VALUE="basketball">Basketball
             <OPTION VALUE="hockey">Hockey
             </SELECT>
      <P>
   <INPUT TYPE="hidden" NAME="recipient" VALUE="sport@domain.com">
   <INPUT TYPE="submit">
   <INPUT TYPE="reset">
   </FORM>
   </BODY>
</HTML>
```

Ilustracja 5.7. *Przykład wiadomości e-mailowej wysłanej przez program FormHandler*

```
From: jas@domena.com
To: <sportowy@sklep.com>
Subject: FormHandler Results

The following information was submitted by
jas@domena.com on Sunday, December 6, 1998 at 21:09:20
----------------------------------------------------------------

sport: football
nazwisko: Jas Kowalski

----------------------------------------------------------------
```

Ilustracja 5.8. *Przykład domyślnej strony potwierdzenia wygenerowanej przez program FormHandler po wypełnieniu formularza*



Ilustracja 5.9 pokazuje zaawansowaną wersję tego samego formularza, zawierającą wymóg wypełnienia pewnych pól, tworzącą plik logu, wysyłającą odwiedzającemu pocztą elektroniczną własną wersję potwierdzenia i wyświetlającą własną wersję strony potwierdzenia. (Sam formularz wygląda jednak dokładnie tak samo jak wersja oryginalna). Różnice w kodzie wytłuszczono, możesz zatem łatwo porównać oba przykłady i zorientować się, co dodaliśmy.

Ilustracja 5.9. *Zaawansowany formularz HTML-owy, który korzysta z prawie wszystkich możliwości programu FormHandler*

```
<html>
  <head>
    <title>Pobranie cennika</title>
  </head>

  <body bgcolor="#ffffff">
    <h1>Pobranie cennika</h1>
```

Ilustracja 5.9. cd. *Zaawansowany formularz HTML-owy, który korzysta z prawie wszystkich możliwości programu FormHandler*

```
    <p>Aby otrzymac cennik naszego sklepu sportowego nalezy
       wprowadzic swoj adres email oraz nazwisko w ponizszym
       formularzu:</p>
    <form action="/cgi-bin/FormHandler.cgi" method=POST>
<pre>
 Nazwisko: <input type="text" name="imie">
 Nazwisko: <input type="text" name="nazwisko">
   E-mail: <input type="text" name="email">

    Dział: <select name="nazwa_cennika">
       <option value="">--Wybierz--
            <option
value="mime:/home/sklep/cennik2.pdf">Baseball
       <option value="mime:/home/sklep/cennik3.pdf">Pilka nozna
            <option
value="mime:/home/sklep/cennik4.pdf">Koszykowka
       <option value="mime:/home/sklep/cennik5.pdf">Hokej
           </select>
</pre>
<br>

      <input type="submit" value="Wyslij">
      <input type="hidden" name="required"
         value="imie,nazwisko,email,sport">
      <input type="hidden" name="field_names"
         value="imie=Imię&Nazwisko=Nazwisko&email=Adres
           Email&nazwa_cennika=Cennik artykułów">
      <input type="hidden" name="sort"
         value="order:imie,nazwisko,email">
      <input type="hidden" name="print_config"
         value="imie,nazwisko,email">
      <input type="hidden" name="success_html_template"
         value="/home/sklep/sukces.html">
      <input type="hidden" name="error_html_template"
         value="/home/sklep/error.html">
      <input type="hidden" name="recipient"
         value="sportowy@sklep.com">
      <input type="hidden" name="cc"
         value="sprzedaz@sklep.com">
      <input type="hidden" name="subject" value="Cennik
        artykułów sportowych">
      <input type="hidden" name="email_template"
         value="/home/sklep/list.txt">
      <input type="hidden" name="reply_message_template"
         value="/home/sklep/reply.txt">
      <input type="hidden" name="reply_message_subject"
         value="Cennik artykułów sportowych">
      <input type="hidden" name="reply_message_from"
         value="sportowy@sklep.com">
      <input type="hidden" name="log_filename"
         value="/home/sklep/log.txt">
      <input type="hidden" name="log_fields"
         value="imie,nazwisko,email,nazwa_cennika,log_date">
      <input type="hidden" name="log_delimiter" value="|">
      <input type="hidden" name="log_uid"
        value="/home/sklep/uid.txt">
    </form>
  </body>
</html>
```

Zachęcamy do budowania formularzy poprzez ich stopniowe rozszerzanie. Zaczynamy od budowy prostych formularzy, korzystając z domyślnych właściwości programu. Następnie dodajemy kolejne opcje, po kolei je testując, by upewnić się, że działają zgodnie z naszymi oczekiwaniami. Kiedy już opanujemy stosowanie *FormHandlera* będziemy mogli pominąć część czynności, ale na początku takie skróty mogłyby powodować zniechęcające Cię błędy w działaniu programu. (*FormHandler* jest prosty w użyciu, ale mnogość obsługiwanych funkcji może z początku powodować pewne zamieszanie).

Tworzenie plików wzorcowych

FormHandler pozwala na stosowanie plików wzorcowych w stosunku do wszystkich generowanych stron oraz listów elektronicznych. Pliki wzorcowe są zwykłymi plikami tekstowymi bądź HTML-owymi, zawierającymi znaczniki zmiennych. Taki znacznik ma następujący format:

```
<<nazwa_zmiennej>>
```

W przypadku *FormHandlera*, *nazwa_zmiennej* odpowiada nazwie pola formularza, zmiennej konfiguracyjnej formularza lub zmiennej środowiskowej CGI. (Odpowiednie tablice zmiennych są przeszukiwane właśnie według takiego porządku). Kiedy *FormHandler* przetwarza wzorzec, każdy znacznik zmiennej zostaje zastąpiony odpowiadającą jej wartością. (Przykłady użycia znaczników można znaleźć w większości plików wzorcowych zawartych w tym rozdziale oraz na ilustracjach 5.10 i 5.11)

Oprócz zmiennych formularza oraz zmiennych konfiguracyjnych omówionych wcześniej, *FormHandler* definiuje dodatkowe zmienne, które można stosować we wzorcach. Stąd wniosek, że ich nazw nie powinniśmy stosować w formularzach.

<<error_fields>> *FormHandler* zmieni ten znacznik na listę pól wymaganych w formularzu, które nie zostały wypełnione przez odwiedzającego. Na przykład:

```
<UL>
  <LI>Your name
  <LI>Your Email Address
</UL>
```

Warto zauważyć, że jeśli zdefiniowaliśmy opisowe nazwy pól, zostaną one tu uwzględnione.

<<date>> *FormHandler* zastąpi ten znacznik wartością bieżącej daty i godziny w odpowiednim formacie (np. '`Tuesday, September 9, 1997 at 12:00:00`').

<<log_date>> *FormHandler* zastąpi ten znacznik wartością bieżącej daty i godziny w specjalnym formacie przeznaczonym do stosowania wewnątrz pliku logu, zależnym od wartości zmiennej log_delimiter (np. '`12:00:00 09/09/97`' lub '`12-00-00 09-09-97`'). Ze zmiennej tej można skorzystać również w pliku wzorca logu lub podając ją jako element zmiennej log_fields formularza, na przykład następująco:

```
<INPUT TYPE="hidden" NAME="log_fields" VALUE="log_date,realname,email">
```

Tworzenie pliku wzorca logu

Wzorzec logu jest najprostszym z wszystkich wzorców, ponieważ w większości przypadków będzie się składał z jednej linii, określającej format pojedynczego rekordu. Na przykład:

```
"<<log_date>>","<<first_name>>","<<last_name>>","<<email>>"
```

Należy się upewnić, że linia jest zakończona znakiem przejścia do nowej linii, w przeciwnym razie kolejny rekord logu będzie doklejany do poprzedniego (w tej samej linii) i trudno nam będzie je rozróżnić. Jeśli użyjemy powyższego wzorca, w logu będą zapisywane linie postaci:

```
"14:40:00 05/05/97","Craig","Patchett","craig@cgi-perl.com"
```

Mimo że w większości przypadków będziemy korzystać z jednoliniowych wzorów, nie ma przeciwwskazań, żeby log zawierał rekordy wieloliniowe, jeśli tego chcemy.

Tworzenie wzorców pocztowych

Wzorce pocztowe są nieco bardziej złożone od wzorców logu, ponieważ oprócz samych pól trzeba wymyślić treść wiadomości, którą chcemy przekazać. Zarówno wzorzec potwierdzenia (które – jak pamiętamy – przesyłane jest do odbiorcy pocztą elektroniczną po wypełnieniu formularza), jak i odpowiedzi (która wysyłana jest do odwiedzającego) są po prostu zwykłymi listami, które mogą zawierać znaczniki zmiennych. W wielu przypadkach nie musimy nawet tworzyć wzorców, o ile domyślny format nam odpowiada (zob. ilustracja 5.8) bądź gdy korzystamy ze zmiennej konfiguracyjnej mail_list do wykonania czynności bardziej złożonej aniżeli wysłanie odpowiedzi (zob. przykład listy dyskusyjnej na końcu rozdziału). Wzorzec odpowiedzi ma trochę inny charakter, gdyż jest wysyłany wyłącznie do odwiedzającego. Zachęcamy do tworzenia własnych wzorców potwierdzenia i odpowiedzi z wykorzystaniem znaczników zmiennych, które pozwalają na przygotowanie listu przeznaczonego dla konkretnej osoby i zawierającego jej imię, nazwisko lub inne dane.

Ilustracja 5.10 pokazuje przykład wzorca odpowiedzi.

Ilustracja 5.10. *Wzorzec odpowiedzi używany do generowania wiadomości Zmienne używane do wstawienia imienia i nazwiska odwiedzającego wytłuszczono*

```
Drogi <<first_name>> <<last_name>>,

Dziekujemy za skorzystanie z naszego serwisu i pobranie cennika
Artykułów sportowych. Dołączyliśmy do niego wybór
najciekawszych Ofert.

Będziemy również Państwa informować o pojawieniu się nowych
Pozycji w cenniku. Jeśli nie chcą Państwo otrzymywać takich
Informacji, prosimy o kontakt.

Pozdrawiam,
Jan Kowalski
Sklep Sportowy
```

Tworzenie wzorca strony potwierdzającej wypełnienie formularza

Wzór strony potwierdzenia to w zasadzie HTML-owa wersja wzorca listu odpowiedzi. Jeśli zdefiniujemy stronę potwierdzenia za pomocą zmiennej success_html_template, będzie ona wyświetlana po wypełnieniu przez odwiedzającego naszego formularza. Za pomocą tej strony można podsumować wprowadzone informacje i podziękować za ich udostępnienie. Można również zachęcić odwiedzającego do zapoznania się z resztą serwisu, bądź zaproponować dalszą drogę poruszania się po nim. (Jedną z głównych zalet własnego wzorca strony potwierdzenia jest możliwość dostosowania jej wyglądu do reszty serwisu). Ilustracja 5.11 pokazuje przykład wzorca strony potwierdzającej, a ilustracja 5.3 – stronę wygenerowaną przez ten wzorzec.

Ilustracja 5.11. *Wzorzec strony potwierdzającej wypełnienie formularza pokazanej na ilustracji 5.3*

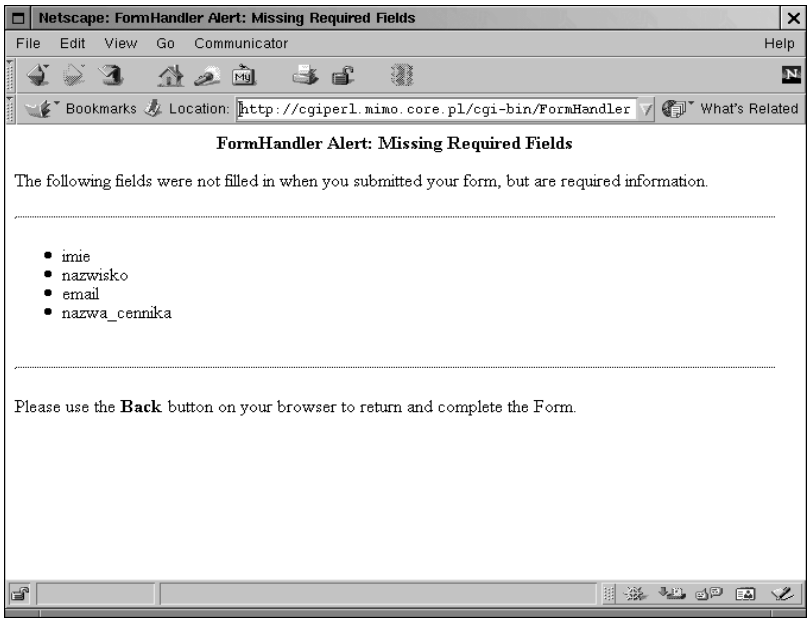
```
<HTML>
   <HEAD>
      <TITLE>Twoje zlecenie zostało zaakceptowane</TITLE>
   </HEAD>
   <BODY BGCOLOR="#EEEEEE">
    <CENTER>
      <H1>Twoje zlecenie zostało zaakceptowane</H1>
    </CENTER>
    <P>
      Dziekujemy za skorzystanie z naszego serwisu.
      Twoje zlecenie pobrania cennika zostało zaakceptowane,
      a odpowiedL wysłana pod adres: <<email>>.
    </P>

    <P>
      Pozdrawiamy,
      </P>
      Sklep Sportowy
   </BODY>
</HTML>
```

Wzorzec strony komunikatu o nie wypełnionych polach

Ostatnim wzorcem, który używany jest w programie, jest wzorzec komunikatu o nie wypełnionych polach. Jak można się domyśleć, służy on do generowania strony, która pojawia się w przypadku, gdy użytkownik nie wypełni wszystkich wymaganych pól. (Podobnie jak dla wzorca strony potwierdzenia, definicja własnego wzorca komunikatu o nie wypełnionych polach jest użyteczna, gdy chcemy dostosować jego wygląd do reszty serwisu). Jak sobie przypominamy, *FormHandler* korzysta ze zmiennej **error_fields**, dzięki której przygotowanie wzorca jest niezwykle proste.

Ilustracja 5.12 pokazuje przykład strony zawierającej listę brakujących pól, która zostanie domyślnie wygenerowana, jeśli nie podamy własnego wzorca. Ilustracja 5.13 zawiera przykład wzorca, a ilustracja 5.2 przykład strony wygenerowanej na jego podstawie.

Ilustracja 5.12. *Domyślna wersja strony komunikatu o nie wypełnionych polach*



Ilustracja 5.13. *Wzorzec komunikatu o nie wypełnionych polach stosowany do wygenerowania strony pokazanej na ilustracji 5.2*

```
<HTML>
   <HEAD>
      <TITLE>Brak pełnych informacji!</TITLE>
   </HEAD>
   <BODY BGCOLOR="#EEEEEE">
      <CENTER>
      <H1>Brak pełnych informacji!</H1>
      </CENTER>
      <P>
      Zapomniałeś wypełnić następujące pola:
      </P>
      <<error_fields>>
      <P>Niestety, nie możemy zrealizować zlecenia, dopóki nie
         zostaną one wypełnione. Cofnij się do poprzedniej
         strony i je uzupełnij.</P>
   </BODY>
</HTML>
```

Dodatek specjalny: lista dyskusyjna

FormHandler jest bardzo elastycznym programem w sensie możliwości korzystania z różnego typu formularzy oraz mnogości zastosowań. Ponadto ma on jedną ciekawą właściwość, która nie jest już tak oczywista (mimo że wspomnieliśmy o niej wcześniej), a może być niezwykle przydatna. Dlatego odejdziemy w tym miejscu od ogólnego schematu budowy tej książki i pokażemy, w jaki sposób *FormHandler* może zostać użyty do tworzenia prostej listy dyskusyjnej (dystrybucyjnej).

Dodawanie osób do listy

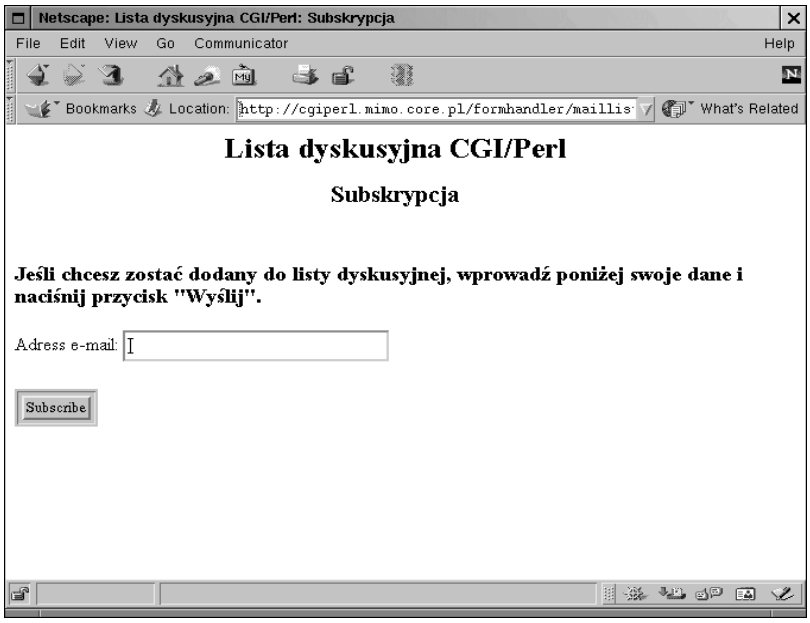
Wysyłanie wiadomości pod adresy z listy dystrybucyjnej jest bardzo proste – wystarczy ustawić odpowiednią zmienną mail_list tak, aby wskazywała plik zawierający listę adresów. Trudniej jest wymyślić prosty sposób dopisywania nowych osób do listy bez konieczności ręcznego ich dopisywania bądź tworzenia osobnego programu. Rozwiązaniem jest niestandardowe wykorzystanie pliku logu do zapamiętywania adresów e-mailowych. Można to zrobić dodając następujące linie do formularza subskrypcji:

```
<INPUT TYPE="hidden" NAME="log_filename"
VALUE="/home/mail_list/list.txt">
<INPUT TYPE="hidden" NAME="log_fields" VALUE="email">
```

Teraz za każdym razem, gdy odwiedzający wpisze swój adres, będzie on dodawany do pliku logu. Ilustracja 5.14. pokazuje kod HTML-owy przykładowego formularza subskrypcji korzystającego z tego triku, zaś ilustracja 5.15 pokazuje, jak taki formularz wygląda.

Ilustracja 5.14. *Formularz używany do subskrypcji przykładowej listy dyskusyjnej*

```
<HTML>
   <HEAD>
      <TITLE>
         Lista dyskusyjna CGI/Perl: Subskrypcja
      </TITLE>
   </HEAD>
   <BODY BGCOLOR="#FFFFFF">
      <CENTER>
         <H1>Lista dyskusyjna CGI/Perl</H1>
         <H2>Subskrypcja</H2>
      </CENTER>
      <BR><BR>
      <H3>Je¶li chcesz zostać dodany do listy dyskusyjnej,
          wprowadŁ poniżej swoje dane i naci¶nij przycisk
          "Wy¶lij".</H3>
      <P>
      <FORM action="http://www.domain.com/cgi-bin/
       formhandler.cgi" method="POST">
         Adress e-mail:
         <INPUT TYPE="text" NAME="email" SIZE=30></H3>
         <INPUT TYPE="submit" NAME="subscribe"
            VALUE="Subscribe">

         <INPUT TYPE="hidden" NAME="required" VALUE="email">
         <INPUT TYPE="hidden" NAME="recipient" VALUE="none">
         <INPUT TYPE="hidden" NAME="field_names"
            VALUE="email=Email Ňaddress">
         <INPUT TYPE="hidden" NAME="log_filename"
            VALUE="/home/web/formhandler/list.txt">
         <INPUT TYPE="hidden" NAME="log_fields" VALUE="email">
         <INPUT TYPE="hidden" NAME="redirect"
            VALUE="http://www.domain.com/formhandler/
             sendmsg.html">
      </FORM>
   </BODY>
</HTML>
```



Ilustracja 5.15. *Przykład formularza wygenerowanego na podstawie kodu z ilustracji 5.14*

Wysyłanie wiadomości na listę

Teraz, kiedy już wiemy w jaki sposób dodawać adresy subskrybentów, wystarczy przygotować kolejny formularz zawierający pola, które chcemy dołączyć do wiadomości wysyłanych na listę (tj. imię i nazwisko, adres poczty elektronicznej oraz wiadomość) oraz następujące pole ukryte:

```
<INPUT TYPE="hidden" NAME="mail_list"
VALUE="file:/home/mail_list/list.txt">
```

W ten sposób ustawiamy zmienną mail_list tak, aby wskazywała na plik logu. Ilustracja 5.16 pokazuje kod HTML-owy przykładowej strony, która to robi, a ilustracja 5.17 pokazuje wygląd działającego formularza.

Tworzenie wzorca wiadomości

Korzystanie z *FormHandlera* w celu budowy listy dyskusyjnej ma swoje wady i zalety. Jedna z głównych zalet jest jednocześnie jedną z większych wad i polega na konieczności użycia formularza do wysyłania listu. Zaleta polega na tym, że taką listę trudno wykorzystać do wysyłania spamu, ponieważ nie ma możliwości wysłania listu przez program pocztowy. Z kolei wadą jest to, że nie można wysłać odpowiedzi na listę korzystając z ulubionego programu pocztowego i trzeba posłużyć się przeglądarką WWW. Tę niedogodność można jednak zredukować dołączając do listów adres formularza. Poza tym wzór listu elektronicznego będzie zazwyczaj bardzo prosty (patrz ilustracja 5.18). Ilustracja 5.19 pokazuje list wygenerowany na podstawie tego wzorca.

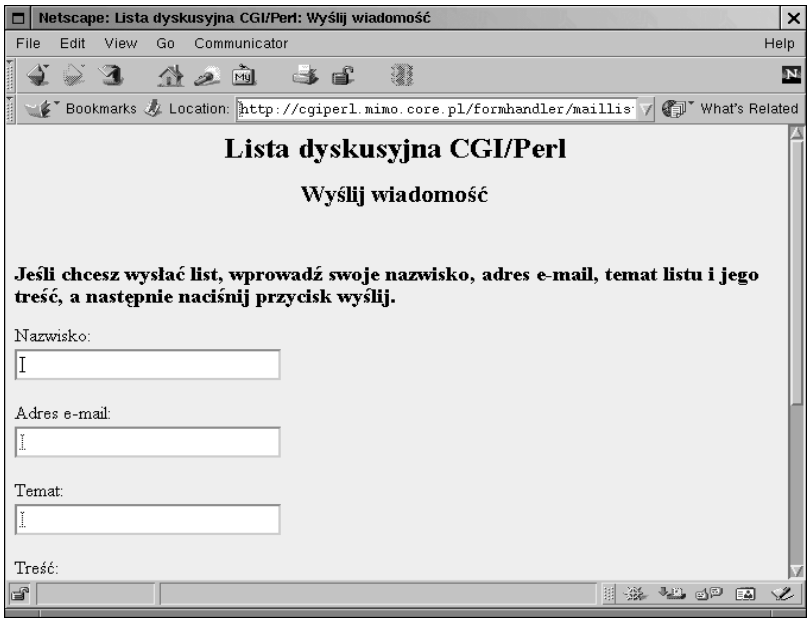
Ilustracja 5.16. *Kod HTML-owy przykładowego formularza wysyłania wiadomości na listę dyskusyjną (linie, z których korzysta FormHandler, wytłuszczono)*

```
<HTML>
   <HEAD>
      <TITLE>
         Lista dyskusyjna CGI/Perl: Wyślij wiadomość
      </TITLE>
   </HEAD>
   <BODY BGCOLOR="#EEEEEE">
      <CENTER>
         <H1>Lista dyskusyjna CGI/Perl</H1>
         <H2>Wyślij wiadomość</H2>
      </CENTER>
      <BR><BR>

      <H3>Jeśli chcesz wysłać list, wprowadź swoje nazwisko,
          adres e-mail, temat listu i jego treść, a następnie
          naciśnij przycisk wyślij.</H3>
      <P>
      <FORM action="http://www.domain.com/cgi-bin/formhandler.cgi"
            method="POST">
         <P>Nazwisko:<BR>
         <INPUT TYPE="text" NAME="realname" VALUE="" SIZE=30>
         <P>
         Adres e-mail:<BR>
         <INPUT TYPE="text" NAME="email" VALUE="" SIZE=30>
         <P>
         Temat:<BR>
         <INPUT TYPE="text" NAME="subject" VALUE="" SIZE=30>
         <P>
         Treść:<BR>
         <TEXTAREA NAME="message" ROWS=10 COLS=70 WRAP="HARD">
         </TEXTAREA>
         </H3>
         <INPUT TYPE="submit" NAME="send" VALUE="Wyslij list">

         <INPUT TYPE="hidden" NAME="required"
            VALUE="realname,email,message">
         <INPUT TYPE="hidden" NAME="recipient" VALUE="none">
         <INPUT TYPE="hidden" NAME="mail_list"
            VALUE="file:/home/web/formhandler/list.txt">
         <INPUT TYPE="hidden" NAME="email_template"
            VALUE="/home/web/formhandler/message.txt">
         <INPUT TYPE="hidden" NAME="field_names"
            VALUE="realname=Name,email=Email Address">
         <INPUT TYPE="hidden" NAME="log_filename"
            VALUE="/home/web/formhandler/messages.txt">
         <INPUT TYPE="hidden" NAME="log_fields"
            VALUE="log_date,realname,email,message">
      </FORM>
   </BODY>
</HTML>
```

Instalacja

Instalacja *FormHandlera* sprowadza się do dwóch kroków. Pierwszy polega na utworzeniu katalogu i instalacji różnych plików wykorzystywanych przez program. Drugi krok polega na zdefiniowaniu zmiennych konfiguracyjnych, które znajdują się w dwóch miejscach – w samym programie i w formularzu, który z niego korzysta.

Ilustracja 5.17. *Przykład formularza wygenerowanego na podstawie kodu HTML-owego z ilustracji 5.16*



Ilustracja 5.18. *Przykładowy wzór listu do wykorzystania w ramach listy dyskusyjnej z ilustracji 5.16 oraz 5.17*

```
Dnia <<date>>, <<realname>> (<<email>>) wysłano następujący list:

<<message>>

------------------------------------------------------------------
Aby odpowiedzieć, skorzystaj z opcji 'odpowiedź' czytnika
poczty.

Jeśli chcesz wysłać odpowiedź na listę dyskusyjną, skorzystaj
z formularza pod adresem: http://www.domain.com/formhandler/
post.html
```

Ilustracja 5.19. *Przykład wiadomości wygenerowanej ze wzorca odpowiedzi z ilustracji 5.18*

```
From: Jan Kowalski (jan@kowalski.com)
To: lista@domena.com
Subject: Podziekowania

Dnia 21/11/98, Jan Kowalski (jan@kowalski.com) wysłać
następujący list:

Dziękuje wszystkim, którzy pomogli mi w instalowaniu programów
z książki o CGI/Perl wydawnictwa Helion.

------------------------------------------------------------------
Aby odpowiedzieć, skorzystaj z opcji 'odpowiedż' czytnika
poczty.

Jeśli chcesz wysłać odpowiedź na listę dyskusyjną, skorzystaj
z formularza pod adresem: http://www.domain.com/formhandler/
post.html
```

Przygotowanie katalogów i plików

Instalacja plików i katalogów programu przebiega następująco:

1. Kopiujemy plik FormHandler.cgi do katalogu cgi-bin naszego serwera i ustawiamy odpowiednie prawa dostępu, aby był traktowany jak program CGI (zob. rozdział 3.).

2. Określamy, gdzie chcemy zainstalować, i ewentualnie tworzymy odpowiednie katalogi dla:

   plików formularzy,

   plików danych (cc, bcc, listy adresowe),

   plików logów,

   plików wzorców,

   tymczasowych plików blokad (używanych do blokowania dostępu do innych plików).

   Ilustracja 5.20 przedstawia przykładową strukturę katalogów.

3. Upewniamy się, że mamy w odpowiednim katalogu przynajmniej jeden formularz oraz pliki wzorców i możemy przetestować program. Należy sprawdzić, czy pliki są dostępne dla serwera WWW.

Po zainstalowaniu wszystkich składników kolejnym krokiem jest ustalenie wartości zmiennych konfiguracyjnych.

Ilustracja 5.20. *Przykładowa struktura katalogów programu FormHandler. Gwiazdką oznaczono pliki opcjonalne. Zmienne konfiguracyjne bez znaku $ wskazują na zmienne formularzy*

Katalog/plik	Prawa dostępu	Zmienna konfiguracyjna
Home		
├─ **temp**	drwxrw-rw- (0766)	**$LOCK_DIR**
└─ **web**	drwxr-xr-x (0755)	
├─form.html	-rw-r--r-- (0644)	
├─**files**	drwxr-xr-x (0755)	
│ └─info.txt*	drwxr-xr-x (0755)	**reply_messsage_attach**
└─**formhandler**	drwxr-xr-x (0755)	
├─bcc*	-rw-r--r-- (0644)	**bcc**
├─cc*	-rw-r--r-- (0644)	**cc**
├─email.txt*	-rw-r--r-- (0644)	**email_template**
├─list.txt*	-rw-r--r-- (0644)	**mail_list**
├─log.txt*	-rw-rw-rw- (0666)	**log_filename**
├─log_template.txt*	-rw-r--r-- (0644)	**log_template**
├─missing.html*	-rw-r--r-- (0644)	**error_html_template**
├─reply.txt*	-rw-r--r-- (0644)	**reply_mesage_template**
└─success.html*	-rw-r--r-- (0644)	**success_html_template**

Ustawienie zmiennych konfiguracyjnych

Zmienne przedstawione poniżej będą pozostawały niezmienione bez względu na to, jaki formularz korzystać będzie z programu. Nie mogą one zostać zmienione bez ingerencji w program.

@REFERERS jest tablicą zawierającą adresy IP lub nazwy serwerów, na których mogą znajdować się formularze odwołujące się do naszego programu (np. `('domena.com', '100.100.100.100')`). Jeśli formularz umieszczony jest na serwerze, który nie jest wymieniony na liście, nie będzie obsługiwany i odwiedzający zobaczy komunikat o błędzie. (Nie ma to nic wspólnego z adresem lub domeną odwiedzającego).

@ALLOWED_ATTACH_DIRS jest tablicą zawierającą ścieżki do katalogów, w których znajdują się pliki wysyłane pocztą przez program (np. `('/users/matt/', '/users/craig/')`). Każda nazwa musi kończyć się ukośnikiem. Jeśli chcemy zezwolić na wysyłanie plików z dowolnego miejsca, możemy ustawić tę tablicę na wartość `('all')`. Jednak ze względów bezpieczeństwa odradzamy takie działanie.

@RESTRICTED_ATTACH_DIRS jest tablicą zawierającą nazwy katalogów, z których pliki nie mogą być pobierane i wysyłane pocztą (np. '`/etc/`'). W przypadku serwerów UNIX-owych tablica powinna zawierać przynajmniej katalog `/etc/`, aby zabezpieczyć się przed „podkradaniem" plików konfiguracyjnych, takich jak na przykład plik /etc/passwd. Pamiętajmy, że tablica ta ma wyższy priorytet od tablicy @ALLOWED_ATTACH_DIRS.

$WEB_SERVER określa aktualną domenę i jest wykorzystywana w procedurze *send_email* (*send_email zdefiniowano w* dołączanym pliku sendmail.pl).

$SMTP_SERVER jest pełną nazwą serwera SMTP, z którego korzystamy aby wysyłać wiadomości pocztowe (np. '`smtp.domena.com`'). Zazwyczaj jest to ta sama nazwa, którą wpisałeś jako serwer poczty wychodzącej w swoim programie pocztowym, ale możesz się upewnić pytając administratora serwera. Jeśli dalej nie wiesz, możesz spróbować dodać człon `smtp.` na początku domeny swojego serwera.

$LOCK_DIR zawiera ścieżkę do katalogu, w którym będą przechowywane tymczasowe pliki powstające w trakcie blokowania wyłączności dostępu do innych plików. Katalog musi się również kończyć znakiem końca katalogu ('/' w przypadku UNIX-a lub Windows, ':' w przypadku MacOS-a).

$MAX_WAIT określa w sekundach maksymalny czas oczekiwania w procedurze *lock* na zwolnienie blokady do pliku.

$REQUIRED_DIR określa pełną lub względną ścieżkę do katalogu, w którym można znaleźć wszystkie wymagane (poleceniem `'require'`) pliki. W większości systemów można je umieścić w tym samym katalogu co program i zostawić zmienną pustą. W innych przypadkach należy określić, gdzie się znajdują.

Program

Formularz jest podstawą wszelkiej interakcji CGI, więc sam *FormHandler* jest szczególnie interesujący jako przykład w nauce CGI. Analizując program linia po linii mamy sposobność dokładniejszego przyjrzenia się jego mechanizmom i nauczenia się korzystania z formularzy jako potężnego i inteligentnego pośrednika między nami a odwiedzającym serwis.

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

Zadziwiające, jak wielu użytkowników serwerów UNIX-owych zapomina o tej linii. Dlatego w dalszej części tej książki będziemy często o niej przypominać. Jest ona niezbędna, jeśli program uruchamiany jest na serwerze UNIX-owym i określa, że program napisany jest w Perlu. (Może zajść potrzeba dostosowania jej do umiejscowienia interpretera Perla w naszym systemie. Zazwyczaj będzie to /usr/local/bin/perl lub /usr/bin/perl). Jeśli korzystamy z tej wersji Perla, która wymaga tej linii, Perl potraktuje ją jako komentarz i zignoruje.

Konfiguracja

Pełne objaśnienie tej części programu znajduje się w części „Ustawienie zmiennych konfiguracyjnych". Dla wygody poniżej umieściliśmy pełną listę zmiennych zdefiniowanych w programie:

```
25  @REFERERS = ('domain.com', '100.100.100.100');
33  @ALLOWED_ATTACH_DIRS = ('all');
40  @RESTRICTED_ATTACH_DIRS = ('/etc/');
49  %CONFIG = ('recipient', '',                 'subject', '',
50             'email', '',                     'realname', '',
51             'first_name', '',                'last_name', '',
52             'field_names', '',               'sort', '',
53             'print_config', '',              'env_report', '',
54             'redirect', '',                  'required', '',
55             'error_html_template', '',       'success_html_template', '',
56             'email_template', '',            'reply_message_template', '',
57             'reply_message_attach', '',      'reply_message_subject', '',
58             'reply_message_from', '',        'log_template', '',
59             'log_filename', '',              'log_fields', '',
60             'log_delimiter', '||',           'log_uid', '',
61             'bcc', '',                       'cc', '',
62             'mail_list', '');
67  $WEB_SERVER = 'domain.com';
72  $SMTP_SERVER = 'smtp.domain.com';
77  $LOCK_DIR = '/tmp/';
82  $MAX_WAIT = 5;
90  $REQUIRE_DIR = 'require';
```

W liniach 49. do 62. inicjowana jest tablica asocjacyjna %CONFIG (zob. „Ustawienie zmiennych konfiguracyjnych". Tablica zawiera nazwy każdego z pól jako klucz oraz – w większości przypadków – pustą wartość. Zmiennym można oczywiście przypisać inne wartości, tak jak to jest zrobione w przypadku zmiennej log_delimiter w linii 60.

Jeśli chcesz zobaczyć, w jaki sposób *parse_form* wykorzystuje te informacje do odseparowania odpowiednich pól, zachęcamy do zapoznania się z objaśnieniami tej funkcji, umieszczonymi w rozdziale 11.

Dołączenie wymaganych plików

Kolejną czynnością jest dołączenie dodatkowych plików, których *FormHandler* używa do różnych typowych zadań.

```
 99   push(@INC, $REQUIRE_DIR) if $REQUIRE_DIR;
```

Powyższa linia dołącza katalog określony przez zmienną **$REQUIRE_DIR** do listy katalogów przeglądanych przez interpreter Perla w poszukiwaniu plików dołączanych do programu. Katalogi te zapamiętywane są w tablicy **@INC**. Zwróćmy uwagę na specjalną formę wyrażenia warunkowego *if*. Działa ono dokładnie tak samo jak zwykłe wyrażenie *if*, ale pozwala umieścić warunek po prawej stronie, co ułatwia czytanie kodu.

```
103   require 'parsform.pl';
104   require 'locksubs.pl';
105   require 'template.pl';
106   require 'base64.pl';
107   require 'uuencode.pl';
108   require 'sendmail.pl';
109   require 'chkemail.pl';
110   require 'formdate.pl';
```

W liniach 103. do 110.dołączamy kod procedur bibliotecznych. Ich szczegółowe objaśnienie znajduje się w rozdziale 11.

Sprawdzenie praw dostępu

Zanim przejdziemy do innych zadań, musimy się upewnić, że program został wywołany przez formularz z uprawnionej domeny.

```
121   $check_referer = "0";
```

Najpierw zerujemy flagę **$check_referer**. Będzie ona zawierać wartość zero (fałsz), o ile domena nie zostanie odnaleziona na liście domen uprawnionych do korzystania z programu.

```
127   if ($ENV{'HTTP_REFERER'} =~ m#http://([^/]+)#) {
128       $hostname = $1;
129   }
130   else {
131       $check_referer = 1;
132   }
```

W dalszym ciągu, w linii 127., sprawdzamy zmienną środowiskową HTTP_REFERER, która zazwyczaj zawiera URL do strony, z której program został wywołany, i wykorzystujemy wyrażenie regularne do wydobycia ze zmiennej adresu serwera. Wyrażenie regularne poszukuje najpierw ciągu `http://`, po którym znajduje się ciąg znaków (+) nie zawierających ukośnika (`[^/]`). Nawiasy określają, że Perl powinien zapamiętać ciąg znaków w specjalnej zmiennej `$1`. Warto również zwrócić uwagę na wariację standardowego wyrażania regularnego, jaką zastosowaliśmy powyżej.

Zwykłe wyrażenie może mieć postać:

```
/http:\/\/([^/]+)/
```

Poprzez dodanie na początku litery *m* mogliśmy zastosować dowolne znaki zamiast ukośników. Dzięki temu wewnątrz wyrażenia nie ma potrzeby poprzedzania ich znakami (\) w celu odróżnienia od separatorów. Ta technika znacznie zwiększa czytelność kodu.

Wróćmy jednak do kodu. Jeśli wyrażenie regularne zostanie dopasowane, innymi słowy – jeśli zawiera prawidłowy URL – wtedy linia 128. ustawia **$hostname** na adres serwera. Jeśli sprawdzenie się nie powiodło, wtedy nie mamy określenia, gdzie znajduje się formularz, z którego nastąpiło odwołanie, a wtedy wątpliwość rozstrzygamy na korzyść odwiedzającego i zakładamy, że nastąpiło ono z serwera, który ma dostęp do programu. Jeśli bezpieczeństwo jest szczególnie ważnym czynnikiem, można to zmienić.

```
138  if ($hostname) {
139      foreach $referer (@REFERERS) {
140          if ($hostname =~ /$referer/i) {
141              $check_referer = 1;
142              last;
143          }
144      }
145  }
```

W linii 138. sprawdzamy, czy zmienna $hostname jest ustawiona. Jeśli tak, wtedy w liniach 139. do 144. wykonywana jest pętla, która sprawdza każdy element tablicy @REFERERS, aby określić, czy domena pasuje do któregoś z nich. Jeśli tak – wartość zmiennej $check_referer ustawiana jest na 1 i pętla zostaje przerwana (linia 142.).

```
149  if (!$check_referer) {
150      &error("$ENV{'HTTP_REFERER'} is not allowed access to this program.");
151  }
```

W linii 149. sprawdzamy, czy zmienna $check_referer jest ustawiona. Jeśli nie – program wyświetla komunikat o błędzie i kończy działanie. Wykorzystujemy do tego procedurę *error* (opisaną na końcu programu).

Przygotowanie danych z formularza

Teraz, kiedy określiliśmy już, czy odwiedzający ma dostęp do programu, możemy przejść dalej i przygotować dane do dalszej obróbki. W tym kroku musimy rozdzielić dane dotyczące poszczególnych pól i – w celu ułatwienia dalszego dostępu do nich – umieścić je w tablicy asocjacyjnej.

Przetwarzanie pól formularza

Pierwszym krokiem jest wywołanie procedury *parse_form* (z pliku parseform.pl), która umieści pola formularza wraz z wartościami w tablicach **%FORM** i %CONFIG.

```
159  if (!&parse_form) {
160      &error($Error_Message);
161  }
```

Jak widać, jest to prosta czynność. W linii 159. wywołujemy *parse_form* i sprawdzamy, czy nie zwróciła błędu. Wszystkie funkcje biblioteczne wykorzystywane w tej książce ustawiają zmienną **$Error_Message** w przypadku wystąpienia problemów, więc jeśli *parse_form* zwraca błąd, linia 160. przesyła $Error_Message do procedury *error* i wyświetla komunikat, po czym program kończy działanie.

Ustawienie nazw pól

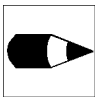
Wszystkie pola są teraz zapisane w tablicach %FORM lub %CONFIG. Następnym krokiem będzie sprawdzenie, czy ustawiona jest zmienna field_names, i przetworzenie jej. (Pamiętajmy, że jest to opcjonalne pole formularza, które zawiera skojarzenie nazw pól z ich rozszerzonymi opisami).

```
169   if ($CONFIG{'field_names'}) {
170       @field_names = split(/&/, $CONFIG{'field_names'});
176       foreach $field_name (@field_names) {
177           ($def_name, $def_value) = split(/=/, $field_name);
178           $ALT_NAME{$def_name} = $def_value;
179       }
180   }
181
```

Jeśli zmienna jest ustawiona, będzie zawierać nazwy pól w następującym formacie:

```
nazwa_pola1=Opis_pola_1&nazwa_pola2=Opis_pola_2
```

Musimy najpierw rozdzielić ją na poszczególne przypisania nazw do pól, a następnie rozdzielić każdą nazwę od opisu, po czym umieścić je w tablicy **%ALT_NAME**. Odbywa się to w liniach 169. do 180. Najpierw, w linii 169., sprawdzamy, czy zmienna field_names jest zdefiniowana. Jeśli tak, korzystamy z funkcji *split* Perla, aby podzielić ją na poszczególne przypisania, które zapisujemy tymczasowo w tablicy **@field_names**. Linia 176. rozpoczyna pętlę wykonywaną dla każdego elementu tej tablicy. Linia 177. korzysta z funkcji *split* do podzielenia każdego przypisania na nazwę i opis pola, a w linii 178. dodajemy odpowiednie wpisy do tablicy %ALT_NAME.



Perl pozwala na stosowanie wyrażenia zwracającego tablicę zawsze tam, gdzie można użyć zmiennej tablicowej (z wyjątkiem lewej strony przypisań). Dlatego linie 170. i 176. można połączyć, otrzymując następujący zapis:

```
foreach $field_name (split(/&/, $CONFIG{'field_names'})) {
```

Sprawdzenie wymaganych pól

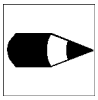
Zanim przejdziemy do przetworzenia danych formularza, musimy się upewnić, że odwiedzający wypełnił wszystkie niezbędne pola. Wśród nich znajdują się pola **email** i **recipient** (należy sprawdzić, czy zawierają poprawne adresy) oraz wszystkie pola wymienione w miennej konfiguracyjnej formularza **required**.

```
194   if (!&email_check($CONFIG{'email'})) {
195       $CONFIG{'email'} = "";
196   }
```

Linia 194. wykorzystuje procedurę *email_check* (z pliku chkemail.pl) do sprawdzenia, czy adres e-mailowy jest we właściwym formacie. (Nie sprawdza jednak, czy on działa i czy możliwe jest dostarczenie poczty pod ten adres). Jeśli nie – w linii 195. kasujemy tę zmienną.

```
201   if (!&email_check($CONFIG{'recipient'}) && $CONFIG{'recipient'}
        !~ /^none$/i) {
202       push(@missing_required_fields, "recipient (invalid address format)");
203   }
```

W linii 201. wykonujemy podobne sprawdzenie w stosunku do pola recipient. Jeśli *email_check* zwróci błąd, sprawdzamy, czy pole ustawione jest na wartość `none` (lub `NONE` – zwróćmy uwagę na literę *i* w wyrażeniu regularnym), która również jest poprawną wartością. Jeśli nie, korzystamy z funkcji *push* Perla do dodania wiadomości do zmiennej tablicowej **@missing_required_fields**, która przechowuje informacje o brakujących polach.



Jeśli korzystamy z wyrażeń regularnych, należy się upewnić, że są one poprawne. Na przykład w linii 201. użyliśmy /^none$/ zamiast /none/, aby mieć pewność, że wyszukiwane będą tylko słowa „none", będące pełną wartością pola, a nie jego częścią (np. mike@danone.com). Jeśli poszukujemy pełnego słowa, można również użyć następującej konstrukcji /\Wnone\W/. (Poszukuje ona słowa „none" otoczonego przez znaki nie należące do zbioru znaków, z których konstruowane są słowa).

```
207   @required = split(/,/, $CONFIG{'required'});
```

Po sprawdzeniu adresu pocztowego możemy przejść do sprawdzenia reszty wymaganych pól (jeśli istnieją), które określone są w zmiennej konfiguracyjnej required. W linii 207. korzystamy z funkcji split do rozdzielenia nazw pól w zmiennej i umieszczenia ich w tablicy **@required**.

```
213   foreach $required (@required) {
214       if (!($CONFIG{$required}) && !($FORM{$required})) {
215           push(@missing_required_fields, $required);
216       }
217   }
```

W linii 213. rozpoczynamy pętlę przeglądającą wszystkie nazwy pól wymienione w tablicy @required. W linii 214. sprawdzamy, czy bieżące pole jest zdefiniowane w tablicy %CONFIG lub %FORM. Jeśli nie, w linii 215. dodajemy jego nazwę do tablicy @missing_required_fields.

```
221   if (@missing_required_fields) {
222       &error('missing_required_fields');
223   }
```

Po zakończeniu sprawdzania wszystkich wymaganych pól, w linii 221. sprawdzamy, czy tablica @missing_required_fields zawiera jakieś elementy. Jeśli tak, w następnej linii wywołujemy procedurę obsługi błędu *error* z komunikatem o brakujących polach.

Inicjalizacja pól daty

Ostatnią czynnością przed właściwym przetwarzaniem formularza jest inicjalizacja dodatkowych zmiennych określających datę, które są używane w różnych częściach programu. Większość z nich korzysta z funkcji *format_date* (znajdującej się w pliku formdate.pl) w celu przetworzenia daty z wewnętrznego formatu Perla na coś bardziej czytelnego. Jest to bardzo poręczna funkcja, dlatego zachęcamy do zapoznania się z jej opisem umieszczonym w rozdziale 11.

```
232   $current_time = time;
```

W linii 232. ustawiamy wartość zmiennej **$current_time** na bieżącą wartość czasu systemowego korzystając z funkcji *time* Perla. W dalszym ciągu będziemy używać jej do ustawiania wartości innych zmiennych.

```
236   $CONFIG{'date'} = &format_date($current_time, "<weekday>, <month> <d>,
                                    <year> at <mtimesec>");
```

W linii 236. korzystamy z procedury *format_date* do ustawienia zmiennej konfiguracyjnej **date**. Jest ona używana w domyślnym formularzu odpowiedzi.

```
242   if ($CONFIG{'log_delimiter'} ne ':' && $CONFIG{'log_delimiter'} ne '/') {
243       $CONFIG{'log_date'} = &format_date($current_time,
                                          "<mtimesec> <0m>/<0d>/<yr>");
244   }
245   else {
246       $CONFIG{'log_date'} = "<mh>-<0n>-<0s> <0m>-<0d>-<yr>";
247   }
```

W liniach 242. do 247. ustawiamy zmienną konfiguracyjną **log_date**, która wykorzystywana będzie podczas tworzenia logów. Linia 242. sprawdza, czy zmienna **log_delimiter** nie zawiera ukośnika lub średnika – jeśli nie można skorzystać z daty w standardowym formacie. Jeśli jednak zawiera ona jeden z tych znaków – wtedy data formatowana jest z użyciem myślników w celu uniknięcia konfliktu z separatorem.

Przetwarzanie zmiennych adresowych

Teraz możemy zacząć przetwarzanie formularza. Pierwszym krokiem jest przetworzenie zmiennych konfiguracyjnych adresu i przystosowanie ich do formatu wymaganego przez procedurę *send_email*.

```
258   if ($CONFIG{'cc'} || $CONFIG{'bcc'} || $CONFIG{'mail_list'}) {
```

Ta część programu wykonywana jest jedynie wtedy, gdy któraś z adresowych zmiennych konfiguracyjnych jest ustawiona.

Przetwarzanie pól CC

Procedura przetwarzania każdej z trzech zmiennych konfiguracyjnych jest taka sama. Dlatego pokażemy ją dokładnie jedynie na przykładzie zmiennej cc, a dla pozostałych po prostu przedstawimy sam kod. (Za pomocą kilku sztuczek moglibyśmy połączyć te trzy części kodu w jedną całość, ale nie byłaby ona wówczas tak czytelna).

```
265     if ($CONFIG{'cc'} =~ /file:(.+)/) {
266         $cc_file = $1;
267         $CONFIG{'cc'} = '';
269         open(CC, $cc_file)
270           || &error("Couldn't open file $cc_file ($!).");
```

W linii 265. sprawdzamy, czy zmienna cc określa plik. Wyrażenie regularne w tej linii poszukuje w niej tekstu `file:`, po którym następuje ciąg znaków (`.+`), który zostanie zapamiętany w zmiennej `$1` (ze względu na umieszczenie nawiasów). Jeśli wyrażenie regularne zostanie dopasowane, wtedy w linii 266. ustawiamy wartość **$cc_file** na nazwę pliku, a w linii 267. usuwamy wartość zmiennej konfiguracyjnej, przygotowując się do pobrania listy adresów z pliku. W liniach 269. i 270. (które tak naprawdę są pojedynczą linią) podejmujemy próbę otwarcia pliku, a w przypadku problemów wywołujemy procedurę *error* z odpowiednim komunikatem.

```
271         while (<CC>) {
272             chop if /\n$/;
273             @cc_addresses = split(/,/, $_);
274             foreach $cc_address (@cc_addresses) {
275                 if (&email_check($cc_address)) {
276                     $CONFIG{'cc'} .= "$cc_address,";
277                 }
278             }
279         }
280         chop $CONFIG{'cc'};
281         close(CC);
282     }
```

Po poprawnym otwarciu pliku w liniach 271. do 281. wczytujemy i przetwarzamy każdą z linii zawartych w pliku. W linii 271. rozpoczynamy pętlę odczytującą linię i przypisującą jej zawartość do zmiennej specjalnej **$_**. (Jest to domyślna zmienna Perla używana w wielu funkcjach). Linia 272. wykorzystuje fakt niejawnego wykorzystywania $_ przez funkcję *chop* oraz porównanie ze wzorcem. W linii tej usuwamy z zawartości zmiennej $_ znak końca linii (jeśli go zawiera). W linii 273. dzielimy linię na pojedyncze adresy e-mailowe i zapamiętujemy je w tablicy **@cc_variables**. Następnie w linii 274. rozpoczynamy pętlę, która przetwarza tablicę i dodaje każdy z adresów do zmiennej konfiguracyjnej cc. (Operator .= służy do dołączenia łańcucha umieszczonego z prawej strony do aktualnej zawartości zmiennej umieszczonej z jego lewej strony). W linii 280., po zakończeniu pętli, usuwamy ostatni przecinek ze zmiennej konfiguracyjnej cc. W linii 281. zamykamy plik, zaś w 282. blok warunkowy *if* rozpoczęty w linii 265.

```
288     elsif ($CONFIG{'cc'}){
289         @cc_addresses = split(/,/, $CONFIG{'cc'});
290         $CONFIG{'cc'} = '';
291         foreach $cc_address (@cc_addresses) {
292             if (&email_check($cc_address)) {
293                 $CONFIG{'cc'} .= "$cc_address,";
294             }
295         }
296         chop $CONFIG{'cc'};
297     }
```

Jeśli zmienna cc nie zawiera nazwy pliku, wtedy w linii 288. sprawdzamy, czy w ogóle coś zawiera, a jeśli tak – przyjmujemy, że jest to adres lub lista adresów oddzielonych przecinkami. W linii 289. rozdzielamy ją na pojedyncze adresy, które zapamiętujemy w zmiennej `@cc_addresses`. Następnie usuwamy dotychczasową wartość zmiennej

konfiguracyjnej cc, aby móc w niej z powrotem zapamiętać tylko ważne adresy. Teraz jesteśmy gotowi do sprawdzenia ważności każdego z nich. W linii 291. rozpoczynamy pętlę przeglądającą każdy z adresów zawartych w @cc_addresses, w linii 292. wywołujemy funkcję *email_check*, która sprawdza ważność każdego z nich i w linii 293. dołączamy rozdzielone przecinkami adresy do zmiennej konfiguracyjnej cc. Po zakończeniu pętli usuwamy ze zmiennej cc ostatni przecinek.

Przetwarzanie pól BCC

Kod przetwarzający pola BCC jest dokładnie taki sam jak w przypadku pól CC, więc dołączamy go poniżej już bez opisu.

```
304      if ($CONFIG{'bcc'} =~ /file:(.+)/) {
305          $bcc_file = $1;
306          $CONFIG{'bcc'} = '';
307
308          open(BCC, $bcc_file)
309            || &error("Couldn't open file $bcc_file ($!).");
310          while (<BCC>) {
311              chop if /\n$/;
312              @bcc_addresses = split(/,/, $_);
313              foreach $bcc_address (@bcc_addresses) {
314                  if (&email_check($bcc_address)) {
315                      $CONFIG{'bcc'} .= "$bcc_address,";
316                  }
317              }
318          }
319          chop $CONFIG{'bcc'};
320          close(BCC);
321      }
327      elsif ($CONFIG{'bcc'}) {
328          @bcc_addresses = split(/,/, $CONFIG{'bcc'});
329          $CONFIG{'bcc'} = '';
330          foreach $bcc_address (@bcc_addresses) {
331              if (&email_check($bcc_address)) {
332                  $CONFIG{'bcc'} .= "$bcc_address,";
333              }
334          }
335          chop $CONFIG{'bcc'};
336      }
```

Przetwarzanie adresów listy dystrybucyjnej

Kod przetwarzający listę dystrybucyjną jest dokładnie taki sam jak w przypadku pola cc, z niewielkim dodatkiem na końcu:

```
343      if ($CONFIG{'mail_list'} =~ /file:(.+)/) {
344          $list_file = $1;
345          $CONFIG{'mail_list'} = '';
346
347          open(LIST, $list_file)
348            || &error("Couldn't open $list_file ($!).");
349          while (<LIST>) {
350              chop if /\n$/;
351              @list_addresses = split(/,/, $_);
```

```
352             foreach $list_address (@list_addresses) {
353                 if (&email_check($list_address)) {
354                     $CONFIG{'mail_list'} .= "$list_address,";
355                 }
356             }
357         }
358         chop $CONFIG{'mail_list'};
359         close(LIST);
360     }
366     elsif ($CONFIG{'mail_list'}) {
367         @list_addresses = split(/,/, $CONFIG{'list'});
368         $CONFIG{'mail_list'} = '';
369         foreach $list_address (@list_addresses) {
370             if (&email_check($list_address)) {
371                 $CONFIG{'mail_list'} .= "$list_address,";
372             }
373         }
374         chop $CONFIG{'mail_list'};
375     }
376 }
```

Linie 343. do 375. są takie same jak 265. do 297. Linia 376. kończy blok warunkowy *if* z linii 258, który sprawdzał, czy są jakiekolwiek adresy do przetworzenia.

Wysyłanie rezultatu przetworzenia formularza

Kiedy mamy już przygotowane wszystkie adresy, możemy wysłać pod nie kopię rezultatu wypełnienia formularza.

Przygotowanie do wysyłki

Zanim zaczniemy wysyłać listy, musimy skonstruować kilka zmiennych.

```
387 if ($CONFIG{'realname'} && $CONFIG{'email'}) {
388     $from = "($CONFIG{'realname'}) $CONFIG{'email'}";
389 }
390 elsif (($CONFIG{'first_name'} || $CONFIG{'last_name'})
      && $CONFIG{'email'}) {
391     $from = "($CONFIG{'first_name'} $CONFIG{'last_name'})
        $CONFIG{'email'}";
392 }
393 elsif ($CONFIG{'email'}) {
394     $from = $CONFIG{'email'};
395 }
396 else {
397     $from = "anonymous\@$ENV{'REMOTE_HOST'}";
398 }
```

W liniach 387. do 398. formatujemy adres nadawcy i zapamiętujemy go w zmiennej **$from**. W linii 387. sprawdzamy, czy zdefiniowano pole **realname** oraz adres e-mailowy nadawcy – jeśli tak, w linii 388. konstruujemy adres nadawcy z obu tych pól. W przeciwnym razie próbujemy w linii 391. skonstruować adres nadawcy w oparciu o zmienne określające imię i nazwisko nadawcy, zapisując wynik w zmiennej $from. Końcowym rezultatem tych operacji jest adres nadawcy w następującej postaci:

```
(Imię Nazwisko) uzytkownik@domena.com
```

Takiego formatu wymaga protokół SMTP, z którego korzysta podczas wysyłania wiadomości procedura *send_email*). Jeśli żadna z tych zmiennych nie została zdefiniowana, w linii 393. Sprawdzamy, czy został podany jakikolwiek adres i jeśli tak, ustawiamy w linii 394. pole $from na ten adres. Wreszcie, jeśli nie dysponujemy nawet tym adresem, konstruujemy anonimowy adres w formacie `anonymous@domena.com`, gdzie zamiast domena.com wystąpi nazwa serwera WWW, na którym uruchomiono program.

```
402  if ($CONFIG{'subject'}) {
403      $subject = $CONFIG{'subject'};
404  }
405  else {
406      $subject = "FormHandler Results";
407  }
```

Ostatnią zmienną, którą musimy ustawić, jest **$subject** zawierająca temat listu. W linii 402. sprawdzamy, czy ustawiona jest odpowiednia zmienna konfiguracyjna formularza i jeśli tak – przepisujemy ją do zmiennej, a jeśli nie – ustawiamy na domyślny tekst.

Przygotowanie domyślnej wiadomości

W dwóch miejscach programu wymagana jest domyślna wiadomość podsumowująca wypełnienie formularza. Pierwsze z nich występuje w sytuacji, gdy nie został wskazany wzorzec listu. Drugie znajduje się w dalszej części programu. Dla ułatwienia obsługi obu scenariuszy przygotujemy już teraz zmienną zawierającą domyślną wiadomość do dalszego użytku.

```
411  if ((!$CONFIG{'email_template'}
412          && ($CONFIG{'recipient'} !~ /^none$/i || $CONFIG{'mail_list'}))
413          && !($CONFIG{'redirect'} || $CONFIG{'success_html_template'})) {
```

Nie ma sensu przygotowywanie wiadomości, która nie będzie użyta, dlatego najpierw upewniamy się, że jest ona potrzebna (w liniach 411. do 413.).

```
414      $results = "The following information was submitted by\n";
415      $results .= "$from on $CONFIG{'date'}\n";
416      $results .= ('-' x 75) . "\n\n";
```

Jeśli przeszliśmy powyższy test, w liniach 414. do 416. zaczynamy przygotowywać wiadomość zapisując wstępny tekst w zmiennej $results. Jeśli linia 416. wydaje się dla kogoś zbyt złożona, to wyjaśniamy, że operator *x* jest stosowany do powtórzenia wcześniej występującego ciągu pewną ilość razy. Ten niezwykle przydatny operator znajdziemy również w dalszych przykładach w książce.

```
421      if ($CONFIG{'print_config'}) {
422          @print_config = split(/,/, $CONFIG{'print_config'});
423          foreach $print_config (@print_config) {
424
425              # If field names have been specified for these config fields,
426              # make sure they get printed into the e-mail.
427
428              if ($CONFIG{$print_config} && $ALT_NAME{$print_config}) {
429                  $results .= "$ALT_NAME{$print_config}:
                        $CONFIG{$print_config}\n";
430              }
431              elsif ($CONFIG{$print_config}) {
432                  $results .= "$print_config: $CONFIG{$print_config}\n";
```

```
433             }
434         }
435         $results .= "\n";
436     }
```

Kolejnym krokiem jest dodanie do wiadomości wszystkich zmiennych konfiguracyjnych, których zażądano za pomocą zmiennej **print_config**. (Jeśli jest ona ustawiona, to zawiera listę zmiennych oddzielonych przecinkami, które należy dołączyć do wiadomości). Linia 421. sprawdza, czy zmienna print_config została ustawiona, jeśli tak, w liniach 422. do 435. aktualizujemy odpowiednio $result. W linii 422. dzielimy zmienną na pojedyncze nazwy pól i zapamiętujemy je w tablicy **@print_config**. W linii 423. rozpoczynamy pętlę, która służy do przetwarzania każdej z nazw oraz ewentualnego ich zastępowania nazwami alternatywnymi z tablicy %ALT_NAME. W linii 429. dołączamy nazwę zmiennej wraz z odpowiadającą jej wartością. W linii 435. dodajemy dodatkowy znak końca linii po zakończeniu pętli (do oddzielenia bloku zmiennych od reszty informacji).

```
440     if ($CONFIG{'sort'} eq 'alphabetic') {
441         @sorted_fields = sort keys %FORM;
442     }
443     elsif ($CONFIG{'sort'} =~ /^order:(.+)/) {
444         @sorted_fields = split(/,/, $1);
445     }
446     else {
447         @sorted_fields = keys %FORM;
448     }
```

Kolejnym krokiem jest dodanie pól formularza do wiadomości, ale zanim przejdziemy do tego kroku, należy je najpierw odpowiednio posortować. Odbywa się to w liniach 440. do 448. Następnie nazwy pól zapamiętywane są w tablicy **@sorted_fields**. W linii 440. sprawdzamy, czy zmienna konfiguracyjna **sort** została ustawiona na sortowanie alfabetyczne. Jeśli tak, w linii 441. sortujemy nazwy pól. Jeśli tak nie jest, sprawdzamy, czy w formularzu nie zdefiniowano już za pomocą zmiennej sort wynikowego porządku pól, wybierając go do zmiennej $1 (za pomocą wyrażenia (.+)) i rozdzielając w linii 444. na poszczególne pola. Jeśli nie określono żadnej z metod sortowania, to w linii 447. przepisujemy do tablic pola w takiej kolejności, w jakiej występują one w tablicy %FORM. (Nie ma żadnej gwarancji co do tego, jaka to jest kolejność, gdyż nie jest ona związana z kolejnością dokonywania wpisów do tej tablicy).

```
452     foreach $sorted_field (@sorted_fields) {
453
454         # Print the name and value pairs in FORM array.
455
456         if ($ALT_NAME{$sorted_field} && $FORM{$sorted_field}) {
457             $results .= "$ALT_NAME{$sorted_field}:
                  $FORM{$sorted_field}\n";
458         }
459         elsif ($FORM{$sorted_field}) {
460             $results .= "$sorted_field: $FORM{$sorted_field}\n";
461         }
462         $SORTED_FIELD{$sorted_field} = 1;
463     }
464     $results .= "\n";
```

Teraz możemy dodać posortowane pola wraz z ich wartościami do zmiennej $results. Linia 452. rozpoczyna pętlę przeglądającą nazwy pól zapamiętanych w tabeli @sorted_fields. W linii 456. sprawdzamy, czy dla przeglądanej zmiennej zdefiniowano nazwę alternatywną

oraz czy zmienna ma zdefiniowaną wartość. Jeśli oba warunki są spełnione, dodajemy do rezultatu nazwę i wartość zmiennej. Jeśli nie, wtedy w linii 459. sprawdzamy, czy zmienna ma zdefiniowaną wartość. Jeśli tak, w linii 460. dodajemy domyślną nazwę zmiennej oraz tę wartość do zmiennej $results. W linii 462. ustawiamy odpowiednie pole w tablicy asocjacyjnej %SORTED_FIELD oznaczające, że dane pole zostało przetworzone. (Najwygodniejszą metodą sprawdzenia obecności klucza jest zastosowanie tablicy asocjacyjnej). W linii 463. kończymy pętlę, a w linii 464. dodajemy do $result dodatkową pustą linię do oddzielenia posortowanych pól od kolejnej sekcji.

```
468     foreach $field_name (keys %FORM) {
469         if (!$SORTED_FIELD{$field_name}) {
470             if ($ALT_NAME{$field_name}) {
471                 $results .= "$ALT_NAME{$field_name}:
                      $FORM{$field_name}\n";
472             }
473             else {
474                 $results .= "$field_name: $FORM{$field_name}\n";
475             }
476         }
477     }
478     $results .= "\n" . ('-' x 75) . "\n\n";
```

Ostatnim krokiem wyprowadzania wartości pól jest sprawdzenie, czy nie pozostały jakieś pola nie uwzględnione w liście pól posortowanych. W liniach 468. do 477. wykonujemy pętlę sprawdzającą, czy każde pole z tabeli %FORM zostało przetworzone. W linii 468. rozpoczynamy pętlę, zaś w następnej przeglądamy tablicę asocjacyjną SORTED_FIELD, aby stwierdzić, czy dane pole zostało już przetworzone. Jeśli nie, sprawdzamy, czy zdefiniowano alternatywną nazwa dla tego pola i dodajemy ją wraz z wartością $result. Jeśli nie zdefiniowano nazwy alternatywnej, po prostu dodajemy nazwę zmiennej wraz z jej wartością. W linii 478. po zakończeniu pętli uzupełniamy $result o dodatkowy separator.

```
482     foreach $env_variable (split(/,/, $CONFIG{'env_report'})) {
483         $results .= "$env_variable: $ENV{$env_variable}\n";
484     }
485 }
```

Ostatnim krokiem przygotowania domyślnej wiadomości jest dodanie zmiennych środowiskowych, określonych w zmiennej konfiguracyjnej **env_report**. Linia 482. dzieli jej zawartość na poszczególne nazwy zmiennych środowiskowych, przechodząc kolejno każdą z nich. (W tej linii skorzystaliśmy z możliwości bezpośredniego przekazywania wyniku jednej funkcji do innej; rezultat wywołania funkcji *split* trafia bez pośrednictwa żadnej tymczasowej zmiennej typu tablicowego do polecenia *foreach*). W linii 485. kończymy rozpoczęty wcześniej (w linii 411.) blok warunkowy.

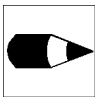
Wysyłanie wiadomości

Jesteśmy teraz gotowi do wysłania wiadomości.

```
489 if ($CONFIG{'email_template'}
      && &valid_directory($CONFIG{'email_template'})) {
490     $message = $CONFIG{'email_template'};
491 }
492 elsif ($CONFIG{'recipient'} !~ /^none$/i || $CONFIG{'mail_list'}) {
493     $message = $results;
```

```
494   }
```

Zdefiniowanie zmiennej konfiguracyjnej **email_template** oznacza, że chcemy skorzystać z własnego pliku wzorca do formatowania wiadomości. W przeciwnym razie zostanie wykorzystany domyślny tekst. W linii 489. sprawdzamy, czy wzorzec został określony i czy znajduje się we właściwym katalogu. Odbywa się to przez wywołanie procedury *valid_directory*, którą zdefiniowano pod koniec programu. Jeśli określono wzorzec, w linii 490. ustawiamy **$message** na nazwę pliku. ($message będzie użyte przez procedurę *send_email* do rozstrzygnięcia, jaki jest format wysyłanej wiadomości). Jeśli nie – w linii 493. ustawiamy jej wartość na domyślny tekst wiadomości.



Jednym ze sposobów na przesłanie kilku tablic do funkcji jest przetworzenie ich na ciągi tekstowe. Podczas wywoływania procedury Perl łączy wszystkie kolejne parametry w jedną tablicę (uniemożliwiając przy tym identyfikację którejkolwiek z tablic przekazanych jako parametr). Bardziej efektywnym, choć przy tym bardziej złożonym sposobem na przekazywanie tablic jako parametrów jest zastosowanie referencji, opisane w większości podręczników Perla.

```
498   if ($CONFIG{'recipient'} !~ /^none$/i) {
499       if (&send_email($subject, $from, $CONFIG{'recipient'}, $CONFIG{'cc'},
500                       $CONFIG{'bcc'}, $message, '', '')) {
501           &error($Error_Message);
502       }
503   }
```

W liniach 498. do 503. przesyłamy wiadomość pod adresy zdefiniowane polem recipient. W linii 498. upewniamy się, że pole to ma wartość różną od none, a następnie wywołujemy procedurę *send_email* próbując wysłać wiadomość. Jeśli wysłanie się nie powiedzie, w linii 501. generowany jest komunikat błędu i program kończy pracę.

```
504   foreach $recipient (split(/,/, $CONFIG{'mail_list'})) {
505       if (&send_email($subject, $from, $recipient, $CONFIG{'cc'},
506                       $CONFIG{'bcc'}, $message, '', '')) {
507           &error($Error_Message);
508       }
509   }
```

W liniach 504. do 509. wysyłamy list na adresy znajdujące się na odpowiednich listach dystrybucyjnych. W linii 504 rozpoczynamy pętlę kolejno przetwarzającą adresy z listy, wywołując w liniach 505. i 506. procedurę *send_email* dla każdego z nich. W przypadku niepowodzenia w linii 507. generujemy komunikat błędu i kończymy pracę programu.

Wysyłanie odpowiedzi

Kolejna część programu jest bardzo podobna do poprzedniej i powinna być łatwa do zrozumienia. Zajmujemy się w niej wysłaniem odpowiedzi do odwiedzającego, o ile taka odpowiedź została określona.

```
516   if (($CONFIG{'reply_message_template'} ||
        $CONFIG{'reply_message_attach'})
```

```
517     && $CONFIG{'email'}) {
```

Odpowiedź chcemy wysyłać jedynie wtedy, gdy odwiedzający udostępnił nam swój adres e-mailowy oraz określono wzór listu lub gdy zachodzi potrzeba wysłania odwiedzającemu pliku. W liniach 516. i 517. sprawdzamy, czy te warunki zostały spełnione.

Przygotowanie do wysyłki

I znów musimy zacząć od skonfigurowania kilku zmiennych pomocniczych.

```
521     $to = $from;
```

W linii 521. wykorzystujemy fakt, że adres nadawcy odpowiedzi jest taki sam jak adres odbiorcy wcześniejszego potwierdzenia wypełnienia formularza.

```
526     if ($CONFIG{'reply_message_from'}) {
527         $from = $CONFIG{'reply_message_from'};
528     }
529     else {
530         $from = $CONFIG{'recipient'};
531     }
```

Teraz możemy ustawić nową wartość zmiennej $from. W linii 526 sprawdzamy, czy została ustawiona wartość zmiennej konfiguracyjnej **reply_message_from**. W takim przypadku ustawiamy wartość stosownie do wartości zmiennej. Jeśli nie – pobieramy wartość ze zmiennej konfiguracyjnej recipient.

```
535     if ($CONFIG{'reply_message_subject'}) {
536         $subject = $CONFIG{'reply_message_subject'};
537     }
538     else {
539         $subject = "FormHandler Reply Message";
540     }
```

Następnie przygotowujemy nagłówek listu podobnie jak poprzednio. W linii 535. sprawdzamy, czy została ustawiona wartość odpowiedniej zmiennej konfiguracyjnej określającej temat listu. Jeśli nie – nadajemy mu temat domyślny.

Przygotowanie plików do wysyłki

Ostatnim krokiem przed wysłaniem listu jest dołączenie do niego wszystkich plików przygotowanych do wysyłki w formacie wymaganym przez procedurę *send_email*.

```
544     if ($CONFIG{'reply_message_attach'}) {
545         local(@attachments) = split(/,/,
              $CONFIG{'reply_message_attach'});
546         foreach $attachment (@attachments) {
547             if ($attachment =~ /^(text|uuencode|mime):(.+)$/) {
548                 $filetype = $1;
549                 $filename = $2;
```

W linii 544. sprawdzamy zawartość zmiennej konfiguracyjnej **reply_message_attach**, aby stwierdzić, czy zawiera listę plików. Jeśli tak – w kolejnych liniach rozdzielamy ją na pojedyncze nazwy plików i zapamiętujemy w tablicy **@attachements**. W linii 546. rozpoczynamy pętlę, w której każdy z nich zostanie odpowiednio przetworzony. W linii 547. upewniamy się, że wybrano właściwą metodę kodowania, zapisując ją w specjalnej

zmiennej $1, zaś nazwę pliku w $2. Jak pamiętamy, właściwym formatem opisu pliku do wysłania jest:

```
text:/ścieżka/do/pliku
uuencode:/ścieżka/do/pliku/do/przekodowania
base64:/ścieżka/do/pliku/do/przekodowania
```

W liniach 548. i 549. zapamiętujemy nazwę pliku w zmiennej **$filename**, a rodzaj kodowania – w **$filetype**. Dzięki temu dalej będziemy mogli porównać je ze wzorcem, które to porównanie mogłoby zniszczyć oryginalne wartości $1 i $2.

```
554                 if (&valid_directory($filename)) {
555                     push(@files, $filename);
556                     if ($filetype eq 'mime') { push(@encoding, 'base64') }
557                     else { push(@encoding, $filetype) }
558                 }
559             }
560         }
```

Zanim dodamy to zlecenie do ostatecznej listy, musimy się upewnić, że plik znajduje się we właściwym katalogu, co stwierdzamy w linii 554. Jeśli tak będzie, nazwa pliku dodawana jest do tablicy **@files**. Następnie sprawdzamy, czy typ kodowania to `mime` i jeśli tak – zmieniamy go na `base64` (Jest to spowodowane metodą komunikacji pomiędzy *send_email* a programem *FormHandler*). W przeciwnym wypadku, w linii 557., zapamiętujemy kodowanie w tablicy **@encoding**.

Linia 559 kończy blok warunkowy z linii 547. zaś linia 560. pętlę *foreach* rozpoczętą w linii 546.

```
565         $file = join(',', @files);
566         $encoding = join(',', @encoding);
567     }
```

Ostatnim krokiem jest połączenie listy plików i ich kodowań w ciągi tekstowe zapamiętane w odpowiednich zmiennych. Stosujemy do tego funkcję *join* Perla. W linii 567. kończymy blok warunkowy *if* rozpoczęty w linii 544., który sprawdza, czy zażądano dołączenia do wiadomości jakiegoś pliku.

Wysyłanie listu

Teraz jesteśmy już gotowi do wysłania odpowiedzi.

```
572     if (&send_email($subject, $from, $to, '', '',
573                     $CONFIG{'reply_message_template'}, $file, $encoding)) {
574         &error($Error_Message);
575     }
576 }
```

Ponieważ wcześniej przygotowaliśmy odpowiedź w formacie wymaganym przez procedurę *send_email*, teraz wystarczy już tylko ją wywołać, co odbywa się w liniach 572. i 573. W przypadku problemów, w linii 574. wywołujemy procedurę *error*. W linii 576. kończy się blok *if* rozpoczęty w linii 516., który określa, czy należy wysłać odpowiedź.

Aktualizacja logu

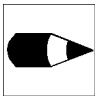
Następnym krokiem przetwarzania formularza jest sprawdzenie i ewentualne zapisanie aktywności użytkowników do pliku logu.

```
584  if ($CONFIG{'log_filename'}
585    && ($CONFIG{'log_fields'} || $CONFIG{'log_template'})) {
```

W celu aktualizacji logu należy ustawić zmienną **log_filename** oraz **log_fields** lub **log_template**. W liniach 584. do 585. sprawdzamy, czy odpowiednie zmienne są ustawione.

```
589      if (!&valid_directory($CONFIG{'log_filename'})) {
590          &error("$CONFIG{'log_filename'} is in a restricted directory.");
591      }
```

W liniach 589. do 591. ponownie wykorzystujemy procedurę *valid_directory* w celu upewnienia się, że log znajduje się w odpowiednim katalogu.



Za każdym razem, kiedy w formularzu korzystamy z plików, należy się upewniać, że dotyczą one plików we właściwym katalogu. W ogólności nie wolno przyjmować założenia co do integralności informacji przesłanych z formularza. Dość łatwe jest skopiowanie formularza i przygotowanie przez użytkownika zmodyfikowanej kopii, która posłuży do wysłania nieprawidłowych danych.

Generowanie unikalnego identyfikatora

Pierwszym krokiem podczas aktualizacji logu jest wygenerowanie unikalnego identyfikatora użytkownika, jeśli jest on potrzebny.

```
597      if ($CONFIG{'log_uid'}) {
601          if (!&valid_directory($CONFIG{'log_uid'})) {
602              &error("$CONFIG{'log_uid'} is in a restricted directory.");
603          }
```

W linii 597. sprawdzamy, czy zdefiniowana jest zmienna konfiguracyjna **log_uid**, która określa, czy należy wygenerować identyfikator. Jeśli tak, w liniach 601. do 603. wywołujemy procedurę *valid_directory*, aby upewnić się, że określony plik znajduje się we właściwym katalogu.

```
608          $log_uid_file = $CONFIG{'log_uid'};
609          $CONFIG{'log_uid'} = '';
```

W linii 608. ustawiamy zmienną **$log_uid_file** na nazwę pliku zapisaną w odpowiedniej zmiennej konfiguracyjnej, a w linii 609. kasujemy zawartość zmiennej log_uid, przygotowując ją do przechowywania identyfikatora użytkownika.

```
613          if (&lock($log_uid_file, $LOCK_DIR, $MAX_WAIT)) {
614              &error($Error_Message);
615          }
```

Możliwe jest, że dwaj różni użytkownicy wyślą formularz w tym samym momencie, w tym przypadku należy się upewnić, że nie otrzymają tego samego identyfikatora. W linii 613.

wywołujemy procedurę *lock* (z dołączonego pliku locksubs.pl) w celu zablokowania pliku identyfikatora. (Zabezpiecza to przed odwołaniami do tego pliku innych programów, które sprawdzą obecność blokady za pomocą procedury *lock*). W linii 614. wywołujemy procedurę *error* w przypadku błędów w blokowaniu.

```
619         if (open(LOG_UID, $log_uid_file)) {
620             $previous_uid = <LOG_UID>;
621             chop($previous_uid) if ($previous_uid =~ /\n$/);
622             $CONFIG{'log_uid'} = ++$previous_uid;
623             close(LOG_UID);
627             open(LOG_UID, ">$log_uid_file")
628               || &error("Can't update $log_uid_file ($!).");
629             print LOG_UID "$CONFIG{'log_uid'}";
630         }
```

W liniach 619. do 629. otwieramy plik identyfikatora, odczytujemy go i aktualizujemy. W linii 619. podejmujemy próbę otwarcia pliku do odczytu. Jeśli próba się powiodła, odczytujemy poprzedni identyfikator i zapamiętujemy go w zmiennej **$previous_uid**. W linii 621. usuwamy z niego zbędny znak końca linii. W linii 622. zwiększamy wartość identyfikatora o jeden i podstawiamy go pod zmienną log_uid, po czym w linii 623. zamykamy plik. W linii 627. podejmujemy próbę otwarcia pliku do zapisu w celu uaktualnienia wartości identyfikatora. W przypadku wystąpienia błędu wywołujemy procedurę *error*. W przeciwnym razie w linii 629. zapisujemy w pliku nowy identyfikator.

```
631         else {
632             open(LOG_UID, ">$log_uid_file")
633               || &error("Can't create $log_uid_file ($!).");
634             print LOG_UID 1;
635             $CONFIG{'log_uid'} = 1;
636         }
637
```

Jeśli w linii 619. wystąpi jakiś problem z otwarciem pliku, przyjmujemy, że go jeszcze nie ma i należy go utworzyć. W linii 632. podejmujemy próbę utworzenia go, a w linii 633. wywołujemy procedurę *error*, jeśli się to nie powiedzie. Następnie zapisujemy do pliku nowy identyfikator.

```
640         close(LOG_UID);
641         &unlock($log_uid_file, $LOCK_DIR);
642     }
```

Bez względu na to czy zaktualizowaliśmy istniejący identyfikator, czy przygotowaliśmy nowy, możemy teraz zamknąć plik i odblokować go. W linii 642. kończymy blok warunkowy rozpoczęty w linii 597.

Zapis rekordu logu

Teraz jesteśmy już gotowi do zapisania rekordu do logu.

```
646     if (&lock($CONFIG{'log_filename'}, $LOCK_DIR, $MAX_WAIT)) {
647         &error($Error_Message);
648     }
649     open(LOG, ">>$CONFIG{'log_filename'}")
650       || &error("Could not open $CONFIG{'log_filename'} ($!).");
```

Linie 646. do 648. blokują plik, a linie 649. do 650. zajmują się jego otwarciem do dopisywania. (Chcemy dołączać nowe rekordy na końcu istniejącego pliku).

```
652     if ($CONFIG{'log_template'} &&
          &valid_directory($CONFIG{'log_template'})) {
653
656         &parse_template($CONFIG{'log_template'}, *LOG)
657             || &error("Could not open $CONFIG{'log_template ($!).'}");
658     }
659     else {
```

W linii 652. sprawdzamy, czy została zdefiniowana zmienna **log_template** i czy wskazuje na plik w dostępnym katalogu, po czym aktualizujemy log wywołując po prostu funkcję *parse_template* i określając nazwę pliku wyjściowego. W przypadku błędów wywołujemy procedurę *error*.

```
664         @log_fields = split(/,/, $CONFIG{'log_fields'});
665         $num_fields = @log_fields;
```

Jeśli nie został określony plik wzorca, należy skorzystać z domyślnego formatu. W linii 644. dzielimy zmienną **log_fields** na indywidualne nazwy pól i zapamiętując je w tablicy **@log_fields**. W linii 665. ustawiamy **$num_fields** na ilość pól logu.

```
667         if ($CONFIG{'log_delimiter'}) {
668             $log_delimiter = $CONFIG{'log_delimiter'};
669         }
670         else {
671             $log_delimiter = \t";
672         }
```

W linii 667. sprawdzamy, czy została zdefiniowana zmienna konfiguracyjna **log_delimieter**. Jeśli tak, to przepisujemy ją do zmiennej **$log_delimiter**. W przypadku gdy nie została zdefiniowana, ustawiamy $log_delimiter na znak tabulacji.

```
676         if ($CONFIG{'log_uid'}) {
677             print LOG "$CONFIG{'log_uid'}$log_delimiter";
678         }
```

Pierwszym polem, które dodamy do pliku logu, jest identyfikator użytkownika (jeśli takowy istnieje). W linii 676. sprawdzamy, czy został on zdefiniowany i jeśli tak, w linii 677. zapisujemy go do logu wraz z odpowiednim separatorem.

```
682         for ($field_num = 0; $field_num < $num_fields; $field_num++) {
683             if ($CONFIG{$log_fields[$field_num]}) {
684                 print LOG $CONFIG{$log_fields[$field_num]};
685             }
686             else {
687                 print LOG $FORM{$log_fields[$field_num]};
688             }
689             print LOG $log_delimiter if ($field_num < $num_fields - 1);
690         }
691         print LOG "\n";
692     }
```

Teraz wystarczy przejrzeć wszystkie wyszczególnione pola logu, zapisując ich wartości do pliku. W linii 682. rozpoczynamy pętlę, która to wykonuje. Następnie sprawdzamy, czy przeglądane pole jest zmienną konfiguracyjną i jeśli nią jest, przepisujemy odpowiednią wartość z tablicy %CONFIG. Jeśli nie – wartość pobieramy z tablicy %FORM.

W linii 689. dodajemy separator, pod warunkiem że nie jest to jeszcze ostatnie pole. Po zakończeniu pętli, w linii 691. dodajemy znak końca linii do oddzielenia rekordu od innych. Linia 692. kończy blok warunkowy *if* rozpoczęty w linii 652.

```
696       close(LOG);
697       &unlock($CONFIG{'log_filename'}, $LOCK_DIR);
698   }
```

Po aktualizacji logu zamykamy plik i zwalniamy jego blokadę. Następnie kończymy blok warunkowy *if* z linii 584.

Wyświetlanie odpowiedzi

Ostatnia rzecz, jaka pozostała do zrobienia, to wyświetlenie podsumowania dla odwiedzającego. *FormHandler* oferuje tutaj kilka możliwości.

Odesłanie do innej strony

Pierwsza możliwość jest najłatwiejsza do zaprogramowania i polega na odesłaniu użytkownika do innej strony.

```
707   if ($CONFIG{'redirect'}) {
711       print "Location: $CONFIG{'redirect'}\n\n";
712   }
```

W linii 707. sprawdzamy, czy ustawiona jest wartość zmiennej **redirect**. Jeśli tak, w linii 711. generujemy nagłówki HTTP, które spowodują, że przeglądarka otworzy inną stronę (więcej informacji na temat nagłówków HTTP znajdziesz w dodatku F).

Wygenerowanie odpowiedzi

Kolejna możliwość to wygenerowanie na podstawie wzorca odpowiedniej strony HTML-owej.

```
713   elsif ($CONFIG{'success_html_template'}
714          && &valid_directory($CONFIG{'success_html_template'})) {
718       print "Content-type: text/html\n\n";
719       if (!&parse_template($CONFIG{'success_html_template'}, *STDOUT)) {
720           &error("Can't open $CONFIG{'success_html_template'} ($!).");
721       }
722   }
```

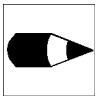
W liniach 713. do 714. sprawdzamy, czy określony został wzorzec odpowiedzi i czy znajduje się on we właściwym katalogu. Jeśli tak, w linii 718. generujemy odpowiedni nagłówek, a w liniach 719. do 721. zajmujemy się przetworzeniem pliku wzorca i obsługą ewentualnych błędów.

Wygenerowanie domyślnej odpowiedzi

Ostatnia możliwość to wygenerowanie domyślnej odpowiedzi programu podsumowującej dane podane w formularzu.

```
723   else {
724
725       # Print a Generic Success HTML Page.
726
727       print <<HTML_END;
728   Content-type: text/html
729
730   <HTML>
731       <HEAD>
732           <TITLE>FormHandler Results</TITLE>
733       </HEAD>
734       <BODY BGCOLOR=#FFFFFF TEXT=#000000>
735          <CENTER>
736          <H1>FormHandler Results</H1>
737          </CENTER>
738          Thank you. Below is a summary of your submission.
739          <P>
740          <PRE>
741   $results
742          </PRE>
743       </BODY>
744   </HTML>
745   HTML_END
746   }
```

Linie 727. do 745. używają wieloliniowego bloku *print* do wygenerowania prostej strony, zawierającej (w linii 741.) rezultat wygenerowany wcześniej (w liniach 414. do 484.). W linii 746. kończymy blok warunkowy *if* z linii 707., który sprawdzał, jakiego typu odpowiedzi udzielić użytkownikowi. Kończymy tam również główny program.



Pusta linia generowana przez 729. linię programu jest niezbędnym elementem formatu nagłówka HTTP. Podobna konstrukcja wystąpiła w linii 711.

Sprawdzanie ważności katalogu (procedura)

W programie występuje kilka miejsc, w których wysyłamy lub przetwarzamy pliki określone w formularzu lub konfiguracji. Jak wspomnieliśmy już wcześniej, z punktu widzenia bezpieczeństwa systemu jest niezwykle ważne, aby upewnić się, że pliki te pochodzą z właściwego katalogu, którego nie dotyczą ograniczenia (ustawiane zwykle przez administratora programu). W tym celu procedura wykorzystuje dwie globalne tablice: **@ALLOWED_ATTACH_DIRS** i **@RESTRICTED_ATTACH_DIRS**.

```
754   sub valid_directory {
```

Linię 754. rozpoczynamy od deklaracji i nazwy procedury.

```
756       local ($filename) = $_[0];
757       local ($allowed_path, $restricted_path);
```

Procedura jest wywoływana z parametrem zawierającym nazwę pliku do sprawdzenia. W linii 756. przypisujemy ją do lokalnej zmiennej **$filename**. Linia 757. deklaruje dwie dodatkowe zmienne lokalne używane wewnątrz procedury.

```
759       local($valid_dir) = 0;
```

Kolejno deklarujemy znacznik **$valid_dir** i inicjujemy jego wartość wpisując zero. Zmienna ta posłuży do śledzenia poprawności lokalizacji pliku w różnych miejscach procedury. Zaczynamy od przypuszczenia, że plik nie znajduje się w prawidłowym katalogu.

```
760     if ($ALLOWED_ATTACH_DIRS[0] =~ /^all$/i) { $valid_dir = 1 }
761     else {
762         foreach $allowed_path (@ALLOWED_ATTACH_DIRS) {
763             $valid_dir = ($filename =~ /^$allowed_path/);
764             last if $valid_dir;
765         }
766     }
```

Najpierw musimy sprawdzić, czy plik znajduje się w dozwolonym katalogu. W linii 760. sprawdzamy, czy pierwszym elementem tablicy @ALLOWED_ATTACH_DIRS jest `all`, w tym przypadku ustawiamy wartość $valid_dir na 1. Jeśli nie – w linii 762. rozpoczynamy pętlę przeglądającą kolejne elementy @ALLOWED_ATTACH_DIRS. Linia 763. jest trochę zagmatwana – służy do sprawdzenia, czy nazwa pliku zawiera na początku ścieżkę z tablicy podstawiając pod $valid_vir wynik porównania (0 lub 1). W linii 764. przerywamy pętlę, jeśli tylko zmienna $valid_dir osiągnęła już wartość 1 (innymi słowy – jeśli odnaleźliśmy już ścieżkę dostępu do pliku w tablicy @ALLOWED_ ATTACH_DIRS).

```
768     foreach $restricted_path (@RESTRICTED_ATTACH_DIRS) {
769         $valid_dir = ($filename !~ /^$restricted_path/);
770         last if !$valid_dir;
771     }
```

Teraz możemy wykonać nieco inny test w odniesieniu do tablicy @RESTRICTED_ ATTACH_DIRS (jeśli takową zdefiniowano). W linii 768. rozpoczynamy pętlę przeglądającą jej elementy. Tym razem sprawdzamy, czy nazwa pliku nie zaczyna się od ścieżki wyszczególnionej w tablicy. Podobnie jak poprzednio, pętla zostaje przerwana, jeśli zmienna $valid_dir zawiera 0. (Warto porównać te linie z liniami 763. i 764. i spróbować odpowiedzieć na pytanie, dlaczego ich działanie jest odwrotne).

```
772     return $valid_dir;
773 }
```

W linii 773. zwracamy ostateczną wartość zmiennej $valid_dir i kończymy blok kodu procedury.

Generowanie komunikatów błędów (procedura)

W niniejszej książce znajdziemy różne sposoby zgłaszania błędów. Większość z nich korzysta z procedury bibliotecznej *error*. Jej wykorzystanie jest oczywiście wyłącznie sprawą własnych preferencji. Program używa lokalnej procedury, a nie procedury z zewnętrznej biblioteki, aby móc elastyczniej obsługiwać różne rodzaje błędów.

```
780 sub error {
```

Procedurę rozpoczynamy od deklaracji jej nazwy.

```
784     local($error) = $_[0];
785     print "Content-type: text/html\n\n";
```

Wywołanie procedury zawiera komunikat błędu jako pierwszy parametr. Przepisujemy go do zmiennej lokalnej **$error** (w linii 785.) i generujemy odpowiedni nagłówek strony.

Raportowanie brakujących pól

W procedurę wbudowano specjalną obsługę błędu dotyczącego brakujących pól.

```
790      if ($error eq 'missing_required_fields') {
795          $CONFIG{'error_fields'} = "<UL>\n";
796          foreach $missing_required_field (@missing_required_fields) {
797              if ($ALT_NAME{$missing_required_field}) {
798                  $CONFIG{'error_fields'} .=
                       "<LI>$ALT_NAME{$missing_required_field}\n";
799              }
800              else {
801                  $CONFIG{'error_fields'} .=
                       "<LI>$missing_required_field\n";
802              }
803          }
804          $CONFIG{'error_fields'} .= "</UL>";
```

W linii 790. sprawdzamy, czy zgłoszono błąd brakujących pól. Jeśli tak, w liniach 795. do 804. ustawiamy zmienną **error_fields** formularza tak, aby zawierała listę brakujących pól. (Skorzystamy z niej w pliku wzorca). Linia 795. rozpoczyna nadawanie wartości zmiennej error_fields znacznikiem HTML-owym otwierającym nie numerowaną listę. W linii 796. rozpoczynamy pętlę przeglądającą poszczególne brakujące pola (które zapamiętane są w zmiennej @missing_required_fields). W linii 797. sprawdzamy, czy dla przeglądanego pola istnieje nazwa alternatywna. Jeśli tak, linia 798. dodaje do error_fields element listy zawierający nazwę alternatywną. Jeśli nie, posługujemy się po prostu nazwą zmiennej. Po zakończeniu pętli dodajemy znacznik HTML-owy zamykający listę.

```
808          if ($CONFIG{'error_html_template'}
809            && &valid_directory($CONFIG{'error_html_template'})) {
810              if (!&parse_template($CONFIG{'error_html_template'},
                   *STDOUT)) {
811                  $error = "Can't open $CONFIG{'error_html_template'}
                       ($!).";
812              }
813              else { exit }
814          }
```

Linie 808. i 809. służą do sprawdzenia, czy został określony wzorzec komunikatu. Jeśli tak, linia 810. podejmuje próbę przetworzenia go, w przypadku niepowodzenia zmieniając odpowiednio komunikat o błędzie. (Teoretycznie moglibyśmy wywołać ponownie procedurę *error*, jednak rozpoczęliśmy już wyświetlanie nagłówka HTML-owego i takie działanie mogłoby wprowadzić w błąd serwer WWW). Jeśli nie ma żadnych dalszych problemów, w linii 813. kończymy program.

```
818          else {
819              print <<HTML_END;
820  <HTML>
821     <HEAD>
822        <TITLE>FormHandler Alert: Missing Required Fields</TITLE>
823     </HEAD>
824     <BODY BGCOLOR=#FFFFFF TEXT=#000000>
```

```
825         <CENTER><H4>FormHandler Alert: Missing Required
              Fields</H4></CENTER>
826         The following fields were not filled in when you submitted your form,
827         but are required information.
828         <P>
829         <HR>
830         <P>
831         $CONFIG{'error_fields'}
832         <BR>
833         <HR>
834         <P>
835         Please use the <B>Back</B> button on your browser to return and
836         complete the Form.
837      </BODY>
838   </HTML>
839   HTML_END
840            exit;
841            }
842        }
```

Jeśli wzorzec komunikatu błędu nie został określony, w dalszej części korzystamy z bloku *print* w celu wygenerowania domyślnej strony komunikatu i kończymy działanie programu. Linia 842. kończy blok warunkowy rozpoczęty w linii 790.

Raportowanie innych błędów

Jeśli błąd nie jest skutkiem niewypełnienia wszystkich wymaganych pól, wtedy generujemy domyślną stronę komunikatu. Zauważmy, że jest ona również wyświetlana w przypadku błędnego przetworzenia wzorca.

```
844        # For any other errors, just print a title and heading which supplies
845        # the error.
846
847        print <<HTML_END;
848   <HTML>
849      <HEAD>
850         <TITLE>$error</TITLE>
851      </HEAD>
852      <BODY BGCOLOR=#FFFFFF TEXT=#000000>
853         <CENTER>
854         <H4>$error</H4>
855         </CENTER>
856      </BODY>
857   </HTML>
858   HTML_END
859        exit;
860   }
```

W liniach 847. do 858. korzystamy z bloku *print*. Linia 859. przerywa program, a linia 860. kończy blok kodu procedury.

Propozycje rozszerzeń

Pomimo że w naszym programie uwzględniliśmy najbardziej poszukiwane właściwości tego typu programów, zawsze jest miejsce na rozbudowę. Jeśli jesteś w twórczym nastroju,

skorzystaj z przedstawionych poniżej propozycji możliwych modyfikacji. (W programie zawarliśmy już większość naszych pomysłów, dlatego poniższa lista jest najkrótsza ze wszystkich publikowanych w tej książce).

Konfiguracja katalogu bazowego. Obecnie *FormHandler* wymaga podawania pełnych ścieżek do plików wzorców, danych itp. Ponieważ większość z nich umieścisz najpewniej we wspólnym katalogu, można by spróbować dodać możliwość określenia katalogu podstawowego w formie zmiennej konfiguracyjnej formularza lub programu (ta druga możliwość jest bezpieczniejsza). Wiele innych programów w tej książce korzysta z różnych katalogów podstawowych, więc w przypadku wątpliwości możesz zawsze spróbować wykorzystać ich kod.

Określenie alternatywnego sposobu sortowania. Określenie porządku, w jakim są wyświetlane oraz zapisywane pola, nie jest zbyt skomplikowane. Można jednakże spróbować uprościć tę czynność dołączając odpowiednią informację bezpośrednio do nazwy pól (co przy okazji zabezpieczy nas przed pominięciem któregoś z pól). Program można łatwo zmienić tak, aby przyjmował pola w następującym formacie:

```
email.P1.L2
```

W ten sposób pole `email` znajdowałoby się pierwsze na wydruku (`P1`), a drugie w logu (`P2`). A może wynajdziesz jeszcze lepszy format?

Kopiowanie plików na serwer. HTML umożliwia tworzenie formularzy do kopiowania plików na serwer. Bez problemów można dodać taką właściwość do *FormHandlera*.

Wieloczęściowe formularze. Czasem potrzeba więcej niż jednego formularza do pobrania informacji od odwiedzającego, szczególnie w przypadku rozbudowanych ankiet. Mimo że jest to nieco bardziej złożone, można bez przeszkód dodać do programu możliwość pracy z wieloczęściowymi formularzami. Cała sztuka polega na przekazywaniu informacji między kolejnymi jego częściami. Można do tego celu użyć ukrytych pól, plików tymczasowych lub cookies. Bardziej zaawansowani użytkownicy mogą spróbować obsłużyć sytuacje, gdy użytkownik zrezygnuje w połowie z dalszego wypełniania formularzy. Zastanów się, jak wówczas aktualizować logi. (Nie jest to wcale takie proste, jak się wydaje).

Administracja logami. Mimo że wymaga to zazwyczaj napisania osobnego programu, byłoby wygodnie mieć możliwość przeglądania i edycji plików logów. Niektóre osoby mogą na przykład wprowadzić nieprawidłowy adres e-mailowy, co spowoduje przesyłanie do administratora komunikatów systemowych o niedostępności adresata poczty. Warto byłoby mieć wtedy możliwość usunięcia odpowiedniego rekordu poprzez interfejs WWW. Dalszą rozbudową byłoby danie użytkownikom możliwości skreślania się z listy dystrybucyjnej.

Ograniczone usługi e-mailowe. Jeśli nie jesteś zainteresowany stosowaniem programu w celu budowy listy dystrybucyjnej, warto poprawić jego system bezpieczeństwa ograniczając adres odbiorcy ($recipient) do jednej lub kilku domen. Na przykład, jeśli serwer zarejestrowany jest w domenie *domena.com*, możesz ograniczyć wartości zmiennej $recipient do takich, które kończą się ciągiem znaków domena.com, blokując w ten sposób możliwość stosowania programu jako bramki pocztowej do wysyłania wiadomości

Twoim serwerem SMTP. (Dotyczy to prawdopodobnie tylko takiej sytuacji, w której program jest wykorzystywany przez wielu użytkowników, ponieważ tablica @REFERERS umożliwia już teraz ograniczanie dostępu).