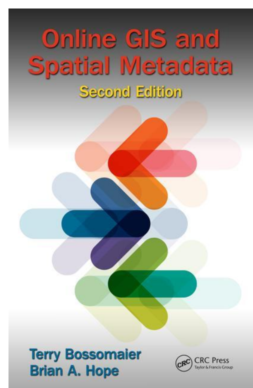
This article was downloaded by: *10.2.98.160*
On: 23 Oct 2021
Access details: *subscription number*
Publisher: *CRC Press*
Informa Ltd Registered in England and Wales Registered Number: 1072954 Registered office: 5 Howick Place, London SW1P 1WG, UK



Online GIS and Spatial Metadata

Terry Bossomaier, Brian A. Hope, Christoph Karon

Server-Side GIS Operations

Publication details
https://test.routledgehandbooks.com/doi/10.1201/b19465-4
Terry Bossomaier, Brian A. Hope, Christoph Karon
Published online on: 18 Dec 2015

How to cite :- Terry Bossomaier, Brian A. Hope, Christoph Karon. 18 Dec 2015, *Server-Side GIS Operations from:* Online GIS and Spatial Metadata CRC Press
Accessed on: 23 Oct 2021
https://test.routledgehandbooks.com/doi/10.1201/b19465-4

PLEASE SCROLL DOWN FOR DOCUMENT

Full terms and conditions of use: https://test.routledgehandbooks.com/legal-notices/terms

This Document PDF may be used for research, teaching and private study purposes. Any substantial or systematic reproductions, re-distribution, re-selling, loan or sub-licensing, systematic supply or distribution in any form to anyone is expressly forbidden.

The publisher does not give any warranty express or implied or make any representation that the contents will be complete or accurate or up to date. The publisher shall not be liable for an loss, actions, claims, proceedings, demand or costs or damages whatsoever or howsoever caused arising directly or indirectly in connection with or arising out of the use of this material.

3

Server-Side GIS Operations

CONTENTS

3.1 Web Servers 39
3.1.1 Popular Web Service Protocols 41
3.1.2 Hypermedia 41
3.2 Server Software 42
3.2.1 Practical Issues 43
3.3 Server Processing 44
3.3.1 CGI and Form Handling 45
3.3.1.1 GET Method 45
3.3.1.2 POST Method 46
3.4 Forms and Image Fields 48
3.4.1 Quality Assurance and Forms 51
3.5 Processing Scripts and Tools 51
3.5.1 Form Processing with Perl 52
3.5.2 Python Scripting Language 54
3.6 Online Map Building 55
3.6.1 The Use of High-Level Scripting Languages 56
3.7 Implementing Geographic Queries 59
3.8 Summary and Outlook 63

Server-Side GIS operations play a critical role in the publication of feature-rich GIS datasets for online access. These operations have been evolving, albeit at a seemingly slower rate than client-side changes. The running of GIS operations on client machines is seen as a way of distributing processing as a form of grid computing. However, much of what happens at the server side is still critical to support the speed and viability of Online GIS systems.

3.1 Web Servers

In this chapter and the one that follows, we examine issues arising in the development and use of standard Web systems as a medium for GIS. Since we can run GIS pack-

Downloaded By: 10.2.98.160 At: 08:01 23 Oct 2021; For: 9781482220162, chapter3, 10.1201/b19465-4

ages, such as ArcInfo, over widespread client server systems such as X-Windows, we might ask what the Web protocol has to offer. And, as we shall see in this chapter, Web-based GIS is not without a few problems. Since the Web server's primary role is to deliver Web pages, server-side GIS operations have to be carried out by secondary programs (usually via the Common Gateway Interface (CGI), which we describe below).

A Web server is a program, or daemon, which runs the HTTP protocol, hence the server is called an HTTP Daemon (HTTPD). The original Web protocol, HTTP, was designed for delivery of static text and simple images, and was not optimised for much else. The gains come in low cost, package independence and in cross-platform Web availability. But any program can package information up and transmit it according to HTTP. This is one strategy available for GIS packages. The other advantage is close integration with a website which may have information and resources going way beyond the spatial. An essential feature of geographic information systems is that they allow the user to interpret geographic data. That is, they incorporate features for data processing of various kinds.

Some of these operations include spatial database queries, map building, geostatistical analysis, spatial modelling and interpolation. In an online GIS, the question immediately arises as to where the above processing is carried out: on the Web server, which is remote from the user and supplies the information, or at the Web client, which receives the information and displays it for the user. In this chapter we look at issues and methods involved in carrying out GIS processing by a Web server. Chapter 4 covers client-side operations.

What GIS operations can and should be run on a server? Only a few operations cannot be run effectively on a server. These consist chiefly of interactive operations, such as drawing, that require rapid response to a user's inputs. Other operations, such as querying large or sensitive databases, must be carried out on the server. However, for most other operations, the question of whether to carry it out on the server or on the client machine is not so clear cut. The two most crucial questions when deciding whether an operation should be performed by the server or the client are

- Is the processing going to place too large a load on the server? Any busy Web server will be accessing and transmitting several files per minute. The processing needed to (say) draw a map may take only a few seconds, but if requests for this operation are being received constantly, then they could quickly add up to an unmanageable load.
- Does the volume of data to be sent to the client place too great a load on the network? For instance, delivering large images constantly might slow response for the user.

Over the last decade, the huge increase in processing speed has made both client-side and server-side processing a lot faster. Broadband speed has also increased for many end users, but there is a caveat – WiFi. Much online GIS now takes place on mobile phones (Chapter 11), where the communication is WiFi, which is often of lower bandwidth. This has led to new metadata specifications (mobileOK, §8.4.2,

Downloaded By: 10.2.98.160 At: 08:01 23 Oct 2021; For: 9781482220162, chapter3, 10.1201/b19465-4

Chapter 11) through which the server can know if it is talking to a mobile device and modify the data transmission accordingly.

3.1.1 Popular Web Service Protocols

The **Hypertext Transfer Protocol** (HTTP) is one of the main communications protocols used on the World Wide Web. It passes hypertext links from a browser to a server and allows the requested documents and images to be passed back to the browser. The server, or HTTP daemon (HTTPD), manages all communication between a client and the programs and data on the website.

Another popular protocol is the **Simple Object Access Protocol** (SOAP). This protocol is a popular communications protocol for exchanging well-structured data between applications over the Internet. Based on XML, SOAP communicates over HTTP and provides a container that can hold a number of different formats of data, making it both simple to use and extensible. One of the advantages of using SOAP is that it can communicate with Web services on different operating systems and programming languages.

Representational State Transfer (REST) is a Web services architecture style that works like the Web. One of its key strengths is that it makes websites usable by machines. It leverages the power of the HTTP application protocol, the URI naming standard and XML. The design of RESTful Web services is simple to use, is versatile and more scaleable than systems designed using Remote Procedure Calls (RPC) (Richardson and Ruby 2007). REST is now used for a significant number of websites where there is a need for machine communication between servers and clients.

Irrespective of the form of Web services implemented, Web services and mashups have now turned the programmable Web into a powerful distributed platform for GIS.

3.1.2 Hypermedia

The World Wide Web has turned the Internet into a medium for hypermedia. The term *hypermedia* is a contraction of hypertext and multimedia. Hypertext refers to text that provides links to other material. *Multimedia* refers to information that combines elements of several media, such as text, images, sound and animation. *Hypertext* is text that is arranged in non-linear fashion. Traditional, printed text is linear: you start at the beginning and read through all the passages in a set order. In contrast, hypertext can provide many different pathways through the material. In general, we can say that hypertext consists of a set of linked objects (text or images). The links define pathways from one text object to another. This is of course the way everyday Web pages are constructed.

Electronic information systems have led to a convergence of what were formerrly very different media into a single form, known as multimedia. Film and sound recordings, for instance, have now become video and audio elements of multimedia publications. In multimedia publications, one or another kind of element tends to dominate. In traditional publications, text tends to be the dominant element, with

Downloaded By: 10.2.98.160 At: 08:01 23 Oct 2021; For: 9781482220162, chapter3, 10.1201/b19465-4

images provided to illustrate the written account. One exception is the comic book, in which a series of cartoon pictures provides the storyline, with text provided as support.

Vision in humans is the dominant sense, so in multimedia, visual elements (particularly video or animation) often dominate. However, in online publications, the speed of the network transmission is still a major consideration for narrow band delivery, such as WiFi. [1] At present full-screen video is not practical, except across the very fastest network connections.

It was the introduction of browsers with multimedia capability that made the World Wide Web a success. However, audio and video elements were not formerly supported by any Web browsers. These require supporting applications that the browser launches when required. Early examples included the programs `mpegplay` for MPEG video files and `showaudio` for sound. Each kind of information requires its own software for generating and editing the material. Now, with the huge popularity of sites such as YouTube, streaming video over the Web is commonplace and is supported by most browsers.

With the advent of HTML5 (§5.3), audio and video built-in applications are now a requisite part of the specificaton. In this chapter we make occasional use of markup formats, distinguished by tags; start tags begin and end with angle brackets as `<map>`; end tags have an additional backslash as in `</map>`. The full syntax and structure of markup for HTML and XML are covered in Chapter 5.

3.2 Server Software

There are many versions of software for Web servers. The most widely used is the Apache server, which is a freeware program, with versions available for all Unix operating systems. However, most of the major software houses have also developed server software. It is not feasible here to give a full account of server software and the issues involved in selecting, installing and maintaining it. Here we can only identify some of the major issues that Web managers need to be aware of. Security is a major concern for any server. Most packages make provision for restricting access to material via authorisation (access from privileged sites) and authentication (user name and password).

With heightened concern over the security of commercial operations, most server software now includes provision for encryption as well as other features to minimise the possibility of illegal access to sensitive information. The rapid rise of online commerce has led to a variant of HTTP, HTTPS, which provides encrypted data transfer, suitable for providing sensitive information such as credit card numbers. In terms of functionality, most server software today makes provision for standard services

[1] WiFi has become so widespread that new standards are continually appearing, each with higher bandwidth than the last. Similarly, telecom companies are offering increasing bandwidth over the mobile phone network.

Downloaded By: 10.2.98.160 At: 08:01 23 Oct 2021; For: 9781482220162, chapter3, 10.1201/b19465-4

such as initiating and running external programs, handling cookies (§4.5), allowing file uploads and making external referrals. Perhaps the most important questions regarding functionality are what versions of HTTP the server software is designed for and how easily upgrades can be obtained and installed. When installing a server, it is important to give careful attention to the structure of the two directory hierarchies. Source material used by HTTP servers normally falls into two main hierarchies:

- The *document hierarchy,* which contains all files that are delivered directly to the client.
- The CGI *hierarchy* (§3.3.1), which contains all the files and source data needed for processing information.

The above distinction is important because it separates freely accessible material, such as documents, from programs and other resources used in processing, which often have security implications. Care is needed too in the organisation of directories under each of these hierarchies. The names of directories normally form part of the URL for any item of information, so it is important that they have logical names, if the URLs are to be referenced directly. However, the widespread use of databases to serve Web pages has led to very long URLs, which are essentially machine readable only.

Moreover, they should reflect the sorts of queries that users will make. Many Web managers make the mistake of structuring directories and information in terms of system management, or in terms of the internal organisation of their institution or corporation. So, for example, users would normally prefer to look for tourist information organised under country or region, rather than (say) the names of individual travel companies or hotel chains. More importantly, the logical names of directories should never change once they are established. It is possible to circumvent this issue to some extent by using aliases to provide a logical hierarchy. In Unix, for instance, directories and files can be assigned logical names that are completely independent of their true storage location. For instance, the real file path

```
/documents/internal-data/file023.dat
```

might be assigned the much simpler logical path

```
hotel-list.
```

3.2.1 Practical Issues

Although not specifically related to online GIS, a number of general issues are important for any online information service. In maintaining a server, three important issues are system updates, backups and server logs. System updates consist of files and data that need to be changed at regular intervals. For example, a data file that is derived from another source may need to be downloaded at regular intervals. These sorts of updates can be automated by using appropriate system software. In the Unix

Downloaded By: 10.2.98.160 At: 08:01 23 Oct 2021; For: 9781482220162, chapter3, 10.1201/b19465-4

Downloaded By: 10.2.98.160 At: 08:01 23 Oct 2021; For: 9781482220162, chapter3, 10.1201/b19465-4

operating system, for instance, the traditional method of automating updates is by setting *crontabs* (a system for setting automated actions against times) to run the necessary shell scripts at regular intervals.

Backups are copies of data on a server. Their function is to ensure that vital data is not lost in case of hardware or other failure. Backups are usually made regularly. For safety, in case of fire or flood, backup copies are best stored off site. Mirror images of data provide another form of backup. However, it may be unwise to rely on an outside organisation to provide the sole form of backup. As with server updates, backups can be automated. Any busy server would need daily backups, though if storage space is limited, these could be confined to copies of new or altered files. However, it is always wise to make a complete copy of the document and CGI hierarchies at regular intervals.

At the time of writing the second edition of this book (2015), the storage world is changing. The cost of storage has fallen dramatically, so much so that many organisations now provide large amounts of storage online, for free, or virtually for free. This has led to *cloud computing*, where data is stored and distributed across multiple servers. Cloud computing (§13.4.1) is still in a growth phase. There are trade-off issues in security, both for and against: the wide distribution of data means that no single catastrophic event will destroy any of the data if it is properly padded with redundancy, but the direct control or hold on the data is sacrificed.

Server log files provide important sources of information about usage of the system. The access log lists every call to the system. As well as recording the names of files or processes that are accessed, it also includes the time and address of the user. This information can be useful when trying to assess usage rates and patterns. The error log is useful in identifying faults in information services, as well as potential attempts to breach system security.

3.3 Server Processing

To be a complete range of media, the Web provides a methodology by which a Web server can carry out processing of various kinds in order to respond to a query. The processing falls into four main classes, which are listed below:

1. Allowing users to submit data to the server, usually via forms;
2. Uploading files to the server;
3. Data processing to derive the information required to answer a query;
4. Building and formatting documents and their elements.

The Common Gateway Interface (CGI) is a link between the Web server and processes that run on the host machine (Figure 3.1). CGI programs are accessed through the Web just as a normal HTML document is, through a URL. The only condition

placed upon CGI programs is that they reside somewhere below the `cgi-bin` directory configured within the HTTP server. This is so the server knows whether to return the file as a document or execute it as a CGI program. The data from a form or query is passed from the client browser to the HTTP server:

1. The HTTP server forwards the information from the browser via CGI to the appropriate application program.
2. The program processes the input and may require access to certain data files residing on the server, such as a database.
3. The program writes either of the following to standard output: an HTML document or a pointer to an HTML document in the form of a Location Header that contains the URL of the document.
4. The HTTP server passes the output from the CGI process back to the client browser as the result of the form or query submission.

Programs to run CGI processes can be written in virtually any programming language. Early examples were PERL, C and shell scripts. PERL was the language of choice for many CGI programmers due to its powerful string manipulation and regular expression matching functionality. This made it well suited to handling CGI input from HTML forms, as well as producing dynamic HTML documents.

The trade-off among these is ease of use and power (such as PERL) and security (compiled languages, such as C). Other languages have now emerged, such as Python and PHP, which allow both client-side and server-side processing. Since this chapter is primarily about concepts and ideas, rather than an application cookbook, we shall stay mostly with a generic language, such as PERL. The basic structure of a CGI program is illustrated by Figure 3.2.

3.3.1 CGI and Form Handling

HTML forms are implemented in a way that produces key-value pairs for each of the input variables. These key-value pairs can be passed to the CGI program using one of two methods, GET and POST. In HTML5 there are two additional methods, PUT and DELETE. However, the uptake of these has been somewhat slow, and we shall not consider them further.

An important point to note is that with both GET and PUT methods, the data is encoded so as to remove spaces and various other characters from the data stream. A CGI program must decode the incoming data before processing.

3.3.1.1 GET Method

The GET method appends the key-value pairs to the URL. A question mark separates the URL proper from the parameters which are extracted by the HTTPD server and passed to the CGI program via the `QUERY_STRING` environment variable. The general format of a GET query is as follows:

```
http://server-url/path?query-string
```

Downloaded By: 10.2.98.160 At: 08:01 23 Oct 2021; For: 9781482220162, chapter3, 10.1201/b19465-4

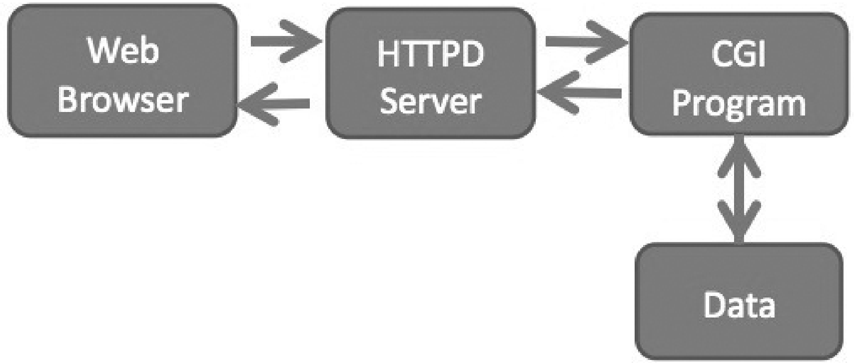


FIGURE 3.1: Role of the Common Gateway Interface (CGI), which mediates exchanges between a Web server and other programs and data.

In this syntax, the main terms are as follows:

server-url is the address of the server that receives the input string;

path is the name and location of the software on the server;

query-string is data to be sent to the server.

Below are some typical examples:

```
http://www.cityofdunedin.com/city/?page=searchtools_st
http://www.geo.ed.ac.uk/scotgaz/scotland.imagemap?30,29
http://www.linz.govt.nz/cgi-bin/place?P=13106
http://ukcc.uky.edu:80/˜atlas/kyatlas?name=Main+&
                                 county=21011
```

The GET method is generally used for processes where the amount of information to be passed to the process is relatively small, as in the above examples. Where larger amounts of data need to be transmitted, the POST method is used.

3.3.1.2 POST Method

In the POST method, the browser packs up the data to be passed to the Web server as a sequence of key-value pairs. When the server receives the data it passes it on to

Downloaded By: 10.2.98.160 At: 08:01 23 Oct 2021; For: 9781482220162, chapter3, 10.1201/b19465-4

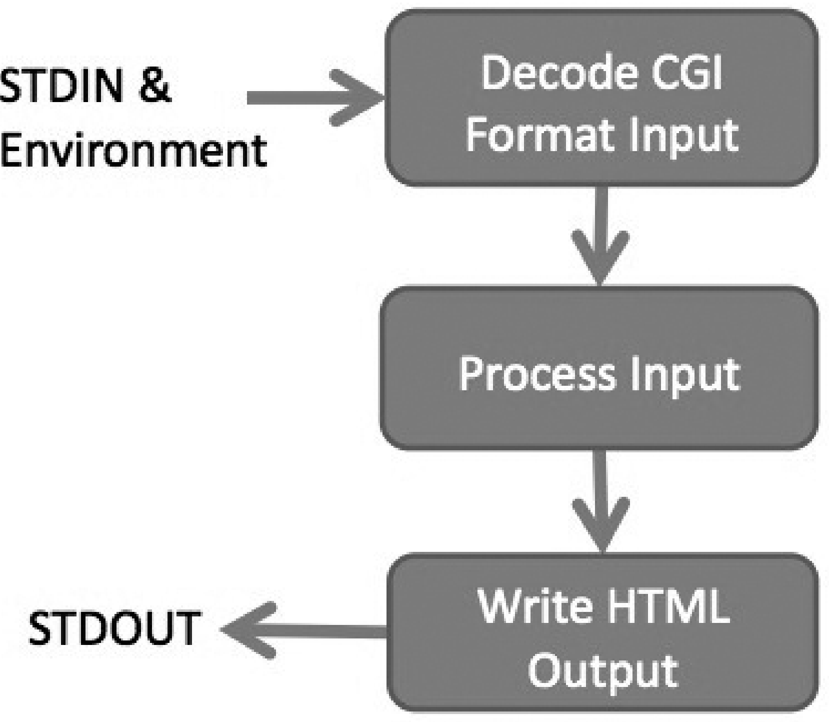


FIGURE 3.2: The structure of CGI programs, where any program must carry out the three tasks shown: first it decodes any form input, then carries out the required processing, and finally it must compile an output document to return to the user.

the CGI program as standard input. Here is a typical string that would be sent by the sample form, which is described in the section that follows.

```
register=tourism&country=Canada&attraction=LaketLouise&
description=&latdeg=Sl&latmin=26&longdeg=-ll6&
longmin=ll&region=Alberta&hotel=YES&meals=YES&park=YBS&
history=YBS&website=http://www.banfflakelouise.com/&
email=info@banfflakelouise.com
```

The application program needs to unpack this string and carve it up into the required name-value pairs before it can use the data.

Downloaded By: 10.2.98.160 At: 08:01 23 Oct 2021; For: 9781482220162, chapter3, 10.1201/b19465-4

TABLE 3.1: Illustrative fields for a Web form where the number of attributes has doubled in HTML5

Field	HTML tag	Attribute	Function
Text	input	text	text entry
Textarea			box for longer text
Select	input	select	pull down list
Radio	input	radio	button selection
Check box	input	checkbox	box to tick
Hidden field	input	hidden	invisible data such as to indentify the form
Submit	input	submit	send form off for processing

3.4 Forms and Image Fields

An important aspect of the Web is that information access need not be passive. Users can send data to Web servers via forms. Forms are documents that include fields where data can be entered. These fields consist of text boxes and various other widgets (Table 3.1), which will be found in any basic book on HTML. Users normally submit form data by clicking on a `SUBMIT` button. The server receives the submitted data, processes it and sends a response back to the user (Figure 3.2).

The basic structure of an HTML form is as follows:

```
<form processing_options>
  input fields mixed with text
</form>
```

Here the *processing options* are attributes that describe the method that will be used to transmit the data (see GET or POST above (§3.3.1)) and indicate which program will receive and process the form data. Table 3.1 lists the types of input fields that can be included. The example below is the HTML source code for a simple form that might be used for (say) operators to register tourist attractions in an online database. The code contains two hidden fields, which provide technical data to the server. One, called "register" tells the server which register to use. This is essential if the same software handles several different services. The second hidden field tells the server which country the registered sites are located in (Canada in this case). This data would be necessary if the underlying database held information for many countries, but the form applied to one only. Note the use of the table syntax to arrange a clear layout of the fields. Web authoring systems usually make it possible to construct HTML documents and forms without seeing the underlying code at all. However, it is not a bad idea for authors to learn to manipulate HTML code directly. For instance, many automatic form builders are features that may not display well on all browsers.

Downloaded By: 10.2.98.160 At: 08:01 23 Oct 2021; For: 9781482220162, chapter3, 10.1201/b19465-4

```
<html>
  <head>
  </head>
  <body>
    <hl>Tourist site register</hl>
    <form
       action="http://life.csu.edu.au/cgi-bin/geo/demo.pl"
       method="POST">
      <input type="hidden" name="register" value="tourism">
      <input type="hidden" name="country" value="Canada">

      <p>Attraction
        <input name="attraction" type="text" size="50"
             value="Enter its name here">

      <p>Description
        <textarea name="description" rows="2"
               cols="40"> Write a brief description here.
        </textarea>

        <h4>Location</h4>
        <table>
          <tr> <td><i>Latitude</i><br>
            <input name="latdeg" type="text" size="3"> deg
            <input name-"latmin" type="text" size="3"> min
               
          <td><i>Longitude</i><br>
            <input name-=longdeg= type="text" size="3"> deg
            <input name="longmin" type="text" size="3"> min

          <td><i>Province</i><br>
            <select name="region">
              <Option value="BC">British Columbia
              <option value="ALB">Alberta
              <option value="SAS">Saskatchewan
              <option value="MAN">Manitoba
              <option value="ONT">Ontario
              <Option value="QUE">Quebec
              <Option value="NB">New Brunswick
              <Option value="PEI">Prince Edward Island
              <option value="NS">Nova Scotia
              <option value="NFL">Newfoundland
            </select>

        </table>

        <h4>Facilities available</h4>

        <table>
```

Downloaded By: 10.2.98.160 At: 08:01 23 Oct 2021; For: 9781482220162, chapter3, 10.1201/b19465-4

```
    <tr><td valign="top"><i>Hotel</i> <td>
     <td><input type="radio" name="hotel" value="YES"> Yes
      <br><input type="radio" name="hotel" value="NO"> No
         
     <td valign="top"><i>Meals</i>
     <td><input type="radio" name="meals" value="YES"> Yes
      <br><input type="radio" name="meals" value="NO"> No
   </table>

   <h4>Features</h4>
   Park <input type="checkbox" name="park" value="TRUE">
   History <input type="checkbox" name="history"
               value="TRUE">

   <h4>Online addresses</h4>
   <table>
    <tr><td><i>Website</i> <br>
      <input name="website" type="text" size="30"
           value="http://">

     <td><i>Email</i> <br>
      <input name==email" type="text= size="20"
           value="Enter.name@location">

   </table>
   <input type="reset" value="Clear">
   <input type="submit" value="Submit">

  </form>
 </body>
</html>
```

Image input fields HTML provides online forms with the capability for images to be used as input fields. That is, if a user points at an image with a mouse and clicks the mouse button, then the location of the mouse pointer will be transmitted as image coordinates. This facility makes it possible to provide maps, within a form, that allow the user to select and submit geographic locations interactively. HTML provides for a user-defined widget called an image field. Although we restrict the discussion here to images that are maps, the image can potentially be anything at all. A typical entry for an image field might be as follows.

```
<input type="image" name="coord" src="world.gif">
```

In this example, the name of the field variable that is defined is coord and the image to be displayed is contained in the file called `world.gif`. The image field so defined has the following important properties.

- The browser transmits not one, but two values from this field. They are the x and y coordinates from the image and are denoted by `coord.x` and `coord.y`, respectively.

Downloaded By: 10.2.98.160 At: 08:01 23 Oct 2021; For: 9781482220162, chapter3, 10.1201/b19465-4

- These coordinates are image coordinates, not geographic coordinates. It is up to the processing software to make the conversion to latitude and longitude.
- The image field acts as a SUBMIT button. That is, when the user clicks on the image, the form data (including the image coordinates) are submitted to the server immediately.

To convert the image coordinates into geographic coordinates, we need to know the size of the image in pixels, and the latitude and longitude corresponding to the two corners. The simplest formula for converting the image coordinate 1 into the corresponding longitude 1 is then

$$L = \frac{x(L_1 - L_0)}{(x_{max} - 1)} + L_0 \tag{3.1}$$

where L_0 and L_1 are the longitudes represented by the top and right sides of the map and x_{max} is the horizontal width of the image in pixels and x begins at zero. A similar formula would apply for extracting latitude from the y coordinate. Note that these formulae apply only on small scales. On a global scale it is necessary to take into account the map projection that is used.

3.4.1 Quality Assurance and Forms

The distributed nature of the Web means that hundreds or even thousands of individuals could potentially contribute data to a single site. This prospect raises the need to standardise inputs as much as possible and to guard against errors. One method, which can be used wherever the range of possible inputs is limited, is to supply the values for all alternatives. So, for instance, instead of inviting users to type in the text for (say) “New South Wales” (i.e., entering it as a text field), we can supply the name as one of several options and record the result in the desired format (e.g., “NSW”), by using a pull-down menu. This method avoids having to sort out the many different ways in which the name of the state could be written (e.g. “NSW”, “N.S.W.”).

3.5 Processing Scripts and Tools

The most basic operation on a server is to return a document to the user. However, many server operations need to include some form of processing as well. For instance, when a user submits a form, the server usually needs to interpret the data in the form and do something with the data (e.g. write it to a file, carry out a search), then write the results into a document that it can return to the user. The processing may include passing the data to various third party programs, such as databases or mapping packages. Some commercial publishing packages now provide facilities to install and manage the entire business. However, the processing itself is often managed using processing scripts. Scripts are short programs that are interpreted by

Downloaded By: 10.2.98.160 At: 08:01 23 Oct 2021; For: 9781482220162, chapter3, 10.1201/b19465-4

the system on the fly. They are usually written in scripting languages, such as Perl (§3.5.1), Python (§3.5.2), Java, Shell Script (Unix/Linux) or Visual Basic (Windows).

3.5.1 Form Processing with Perl

Perl was very popular in the early days of the WWW. It is an extremely powerful text processor, with many arcane shortcuts and extensive code libraries. It was, however, very much a product of the heyday of Unix, with its laconic and at times slightly obfuscating, style. But it was the potential security risks which hastened its decline for WWW applications.

The following short example shows a simple Perl script that processes form data submitted to a Web server. The code consists of three parts. Part 1 decodes the form data (which is transmitted as an encoded string), and the data are stored in an array called list. This array, which is of a type known as an associative array, uses the names of the fields to index the values entered. So for the field called `country` in the form example earlier (which took the value Canada), we would have an array entry as follows: `list{country}=Canada`. Part 2 writes the data in tagged format to a file on the server. Part 3 writes a simple document (which echoes the submitted values) to acknowledge receipt to the user. A detailed explanation of the syntax is beyond the scope of our discussion. There are also many online libraries of useful scripts, tools and tutorials.

```
#!usr/local/bin/perl
#  Simple form interpreter
#  Author: David G. Green, Charles Sturt University
#  Date : 21/12/1994
#  Copyright 1994 David G. Green
#  Warning: This is a prototype of limited functionality.
#          Use at your own risk. No liability will be
#          accepted for errors/inappropriate use.
#
# PART 1 - Convert and store the form data
# Create an associative list to store the data
%list = ();
# Read the form input string from standard input
$command_string = <STDIN>;
chop($command_string);
# Convert codes back to original format
# ... pluses to spaces
$command_string =~ s/\+/ /g;
# ... HEX to alphanumeric
$command_string =~ s/%(..)/pack("c",hex($1))/ge;
# now identify the terms in the input string
$no_of_terms = split(/&/,$command_string);
@word = @_;
# Separate and store field values, indexed by names
for ($ii=0; $ii<$no_of_terms; $ii++)
{     @xxx = split(/=/,$word[$ii]);
```

Downloaded By: 10.2.98.160 At: 08:01 23 Oct 2021; For: 9781482220162, chapter3, 10.1201/b19465-4

```
      $list{$xxx[0]} = $xxx[1];
}
#
# PART 2 - Print the fields to a file in SGML format
$target_name = "formdata.sgl";
open(TARGET,">>$target_name");
# Use the tag <record> as a record delimiter
print TARGET "<record>\n";
# Cycle through all the fields
# Print format <fieldname>value</fieldname>
foreach $aaa (keys(%list))
{     print TARGET "<$aaa>$list{$aaa}<\/$aaa>\n";
}
print TARGET "<\/record>\n";
close(TARGET);
#
# PART 3 - Send a reply to the user
# Writes output to standard output
# The next line ensures that output is treated as HTML
print "Content-type: text/html\n\n";
# The following lines hard code an HTML document
print "<HTML>\n<HEAD>\n Form data
return\n<\/HEAD>\n<BODY>\n";
print "<H1>Form received.</H1>\n<P>Here is the data you
entered ...\n";
# Print the fields in the form FIELD = VALUE
foreach $aaa (keys(%list))
{     print "Field $aaa = $list{$aaa}\n";
}
print "<\/BODY><\/HTML>\n";
```

To understand how this script would be used in practice, suppose that it is stored in an executable file named *simple.pl* that is located in the *cgi-bin* hierarchy of a server whose address is *mapmoney.com*. Then the script would be called placing the following action command in the form:

```
<form action="http://mapmoney.com/cgi-bin/simple.pl"
      method="POST">
```

The purpose of the above example is to show the exact code that can be used to process a form. However, in general, it is not good practice to write scripts that hard-wire details such as the name of the storage file or the text to be used in the acknowledgment. Instead, the script can be made much more widely useful by reading in these details from a file. For instance, the return document can be built by taking a document template and substituting details supplied with the form for the blanks left in the template, or by the script itself. Likewise, the name of the output file could be supplied as a run-time argument to the script. To do this the script would need to replace the line:

```
$target_name  "formdata.sgl";
```

Downloaded By: 10.2.98.160 At: 08:01 23 Oct 2021; For: 9781482220162, chapter3, 10.1201/b19465-4

with an assignment such as the following
`$target_name =@ARGV;`
The URL to call the script would use a "?" to indicate a run-time argument:

```
http://mapmoney.com/cgi-bin/simple.pl?formdata.sgl
```

This example still has problems. In particular, this example shows the extensions of both the script and the exact name of the storage file. For security reasons, it is advisable to avoid showing too many details.

3.5.2 Python Scripting Language

As the Perl star has waned, the Python (from Monty Python's Flying Circus) has risen. Python is an open-source scripting langage that is gaining in popularity over more traditional scripting languages, such as Perl. Python runs on all major operating systems and has implementations for C (CPython), Java (JPython) and C# (IronPython) programming languages. More importantly, Python is considered by many as an ideal programming language for beginners, because of its concise code. Unlike Perl, the Python language is small and must be written in a particular way, the lack of shortcuts making it easier for people to follow code examples when learning the language.

Python can be used in Web applications, scripts and standalone applications, and is a powerful language for both server-side processing and for client systems. Python functions have one or more arguements and a return value of type PyObject*.

The following Python code sample is the equivalent of the Forms Interpretor Perl script in §§3.5.1.

```
#!/usr/bin/python
import cgi

# PART 1 - Get the form variables
form = cgi.FieldStorage()
print form;

# PART 2 - Write the form values to an SGML file
with open( '/tmp/formdata.sgl', 'a' ) as sgmlFile:
   sgmlFile.write( '<record>\n' )
   for key in sorted( form.keys() ):
      sgmlFile.write( '<%s>%s</%s>\n' % ( key, form.getvalue(
          key), key ) )
   sgmlFile.write( '</record>\n' )

# PART 3 - Send the results back to the browser depicting the
    input fields and their values
print 'Content-type: text/html\n\n'
print '<html>\n<head>\n<title>Form data returned</title>\n</
    head>\n<body>\n'
```

Downloaded By: 10.2.98.160 At: 08:01 23 Oct 2021; For: 9781482220162, chapter3, 10.1201/b19465-4

```
for key in sorted( form.keys() ):
   print 'Field %s = %s<br />' % ( key, form.getvalue(key) )
print '\n</body>\n</html>\n'
```

There is an active Python community online[2] where developers can get involved, with a lot of guides, downloads and installations guides all available online for programmers and non-programmers alike.

Python contains a repository of software tools that can be used on Linux, Mac OS X and Windows platforms. Known as the Python Package Index (PyPI), the index contained almost 60,000 packages from third-party development communities[3] at the time of writing. These packages are provided in a number of different stages across the standard development cycle (planning to retired (Inactive)) and can be used under 1 of 13 different licensing models from Creative Commons 1.0 to Proprietary License, Public Domain and freeware.

There is now a variety of spatial-specific Python projects, including projects that have implemented spatial joins and RTree indexing[4] available for use in other applications.

3.6 Online Map Building

To build a simple map across the Web, the following sequence of steps must take place:

1. The user needs to select or specify the details of the map, such as the limits of its borders and the projection to be used;
2. The user's browser (the client) needs to transmit these details to the server;
3. The server needs to interpret the request;
4. The server needs to access the relevant geographic data;
5. The server needs to build a map and turn it into an image (e.g., PNG format);
6. The server needs to build an HTML document and embed the above image in it;
7. The server needs to return the above document and image to the client;
8. The browser needs to display the document and image for the user.

This process is illustrated in Figure 3.3. In the above sequence, only Steps 2, 7 and 8 are standard operations. The rest need to be defined. In almost all cases,

[2] `https://www.python.org/`
[3] `https://pip.pypa.io/en/latest/installing.html`
[4] RTree is a tree data structure which groups and indexes nearby objects through holding a parent extent at the next higher level of the tree, which enables fast spatial operations.

Downloaded By: 10.2.98.160 At: 08:01 23 Oct 2021; For: 9781482220162, chapter3, 10.1201/b19465-4

Step 1 involves the use of a form, which the browser encodes and transmits. In Step 3, the server passes the form data to an application program, which must interpret the form data. The same program must also manage the next three steps: communicating with the geographic data (Step 4), arranging the map building (Step 5) and creating a document to return to the user (Step 6).

When building a map in the above example, what the system actually produces is a text document (which includes form fields) with the map inserted. The map itself is returned as a bit-map (pixel-based) image. The image is in some format that a Web browser can display. Until recently, this usually meant a GIF format, or even JPEG. Having to convert vector GIS data to a pixel image has been a severe drawback. The output loses precision. It cannot be scaled, and downloading a pixel image, even a compressed one, often requires an order of magnitude more bandwidth than the original data. Scalable Vector Graphics (SVG) are a much better solution.

But one new option is the canvas element in HTML5 which provides just such vector operations (§5.3).

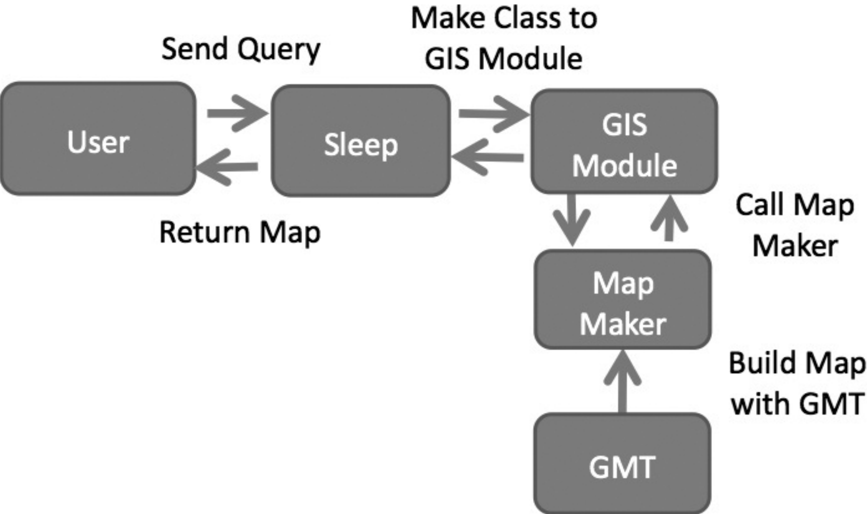


FIGURE 3.3: The flow of information from user to server and back that is involved in a typical system for building maps over the Web. The modules used here are as follows: SLEEP was a script interpreter with a module of GIS calls to the Mapmaker software (Steinke et al. 1996) which uses the GMT package of free map drawing tools (Wessel and Smith 1991, 1995).

3.6.1 The Use of High-Level Scripting Languages

Programming languages were invented to simplify the task of programming computers. High-level languages are computer languages that are designed to simplify programming in a particular context. It is far easier to write a program that carries

Downloaded By: 10.2.98.160 At: 08:01 23 Oct 2021; For: 9781482220162, chapter3, 10.1201/b19465-4

out a specialised task if you can use terms and concepts that relate directly to the system concerned. The problem with general purpose languages is that the solution to a programming problem has to be expressed in terms that are very far removed from the problem's context. In particular, most automating of online services has involved writing programs in languages such as Perl, Java, C++ or shell script.

It has become commonplace in many computing packages to simplify the specification of processing steps by providing high-level scripting languages. The advantages are that most operations can be programmed far more concisely than general purpose languages. Also, because they are oriented towards a specific content area, they are usually easier to learn and to use. For example, to extract the contents of a Web form in the language Perl requires a program of at least a dozen lines. However, in a Web publishing language the entire process is encapsulated in a single command. High-level languages are desirable in developing server-side operations on the Web. The advantages (Green 1996, 2000; Green et al. 1998) include modularity, reusability and efficiency. As we shall see below and in later chapters, high-level languages can include GIS functions and operations. Most conventional GIS systems incorporate scripting languages to allow processes to be automated. In automating a website, particular scripts can be generalised to turn them into general purpose functions. To do this we start with a working script such as the following simple example:

```
SET BOUNDS 34.8S 140.1E 40.4S 145.2E
EXTRACT roads, topography,vegetation
PLOT roads
PLOT topography
PLOT vegetation
```

We then replace constant values by variables, which can be denoted by angle brackets.

```
SET BOUNDS <tlat> <tlong> <blat> <blong>
 EXTRACT roads, topography,vegetation
PLOT roads
PLOT topography
PLOT vegetation
```

This generalised script can now serve as a template for producing a plot of the same kind within any region that we care to select. If the user provides the boundaries from (say) a form, then we could generate a new script by using a Perl script to replace the variables in the template with the new values. Here's a simple example of a Perl script that does this.

```
#!/usr/bin/perl
# Build a simple script from a template
# The associative array markup contains
# replacement values
getvarsfromform;
```

Downloaded By: 10.2.98.160 At: 08:01 23 Oct 2021; For: 9781482220162, chapter3, 10.1201/b19465-4

```
filtertemplate;

sub filtertemplate {
while ($sourceline=<STDIN>)
{ chop($sourceline);
$targetline = $sourceline;
# Enter the input string into the template fields
for ($i=0; $i<$no_of_tags; $i++)
{ $work = $tag[$i];
$targetline =~ s/$work/$formvar{$work}/gi;
}
print "$targetline\n";
}
}
```

This script acts as a filter. Its function is similar to a merge operation in a word processor. We supply values for variables in a form. The function *getvarsfromform* (cf. the example in §3.5) retrieves these values as a table ($formvar). The Perl script then reads in the template as a filter and prints out the resulting script. To run this script we would use a call such as:

```
cat templatefile | filterfile > outputscript
```

where *templatefile* is the file containing the template, *filterfile* is the file containing the above Perl code and *outputscript* is the Perl resulting script.

Although the above procedure works fine, it is cumbersome to have to rewrite scripts for each new application. A more robust and efficient approach is to continue the generalisation process to include the Perl scripts themselves. This idea leads quickly to the notion of implementing Web operations via a high-level publishing language. The following example of output code shows what form data might take after processing by a script such as the above. The format used here is XML (§5) with tags such as `country` corresponding to form fields.

```
<country>
  <name>United State of America</name>
  <info>http://www.usia.gov/usa/usa.htm/</info>
  <www>http://vlib.stanford.edu/Servers.html</www>
  <government>http://www.fie.com/www/us_gov.htm</government>
  <chiefs>http://www.whitehouse.gov/WH/html/handbook.html
  </chiefs>
  <flag>http://www.worldofflags.com/</flag>
  <map>http://www.vtourist.com/webmap/na.htm</map>
  <spdom>
    <bounding>
      <northbc>49</northbc>
      <southbc>25</southbc>
      <eastbc>-68</eastbc>
      <westbc>-125</westbc>
```

Downloaded By: 10.2.98.160 At: 08:01 23 Oct 2021; For: 9781482220162, chapter3, 10.1201/b19465-4

```
    </bounding>
  </spdom>
  <tourist>http://www.vtourist.com/webmap/na.htm</tourist>
  <cities>http://city.net/countries/united_states/</cities>
  <facts>http://www.odci.gov/cia/publications/
  ...factbook/us.html</facts>
  <weather>http://www.awc-kc.noaa.gov/</weather>
  <creator>David G. Green</creator>
  <cid>na</cid>
  <cdate>1-07-1998</cdate>
</country>
```

A simple publishing script converts the above data from XML format (held in the file usa.xml) into an HTML document (stored in the file usa.html). In this case it would replace the tags with appropriate code. The reason for doing this will become clearer in Chapter 5.

Such a script would be easily prepared by a naive user, without understanding what the conversion operations actually are. Many browsers in 2015 are XML aware and they now use stylesheets to do these conversions; however, authoring a stylesheet is still a task for a web developer specialist.

3.7 Implementing Geographic Queries

A simple example of a geographic query online, is the following demonstration of computing great circle distances between points, selected from a map of the world.

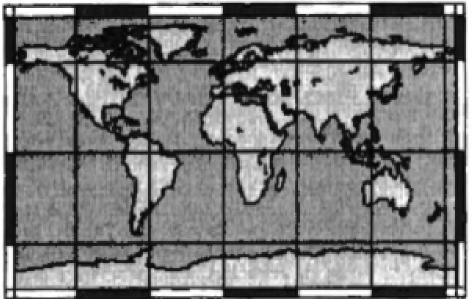
The interface shown in Figure 3.4, is a simple HTML form containing a map image, descriptive text and several data fields. With this form, the user simply clicks on the image map to select a new location for position A or B. A radio button (SELECT POINT) defines which point is being selected. Following each selection, the server regenerates the form with the new values, and the great circle distance is displayed in the box at the right of the map.

To call the example, the URL addresses the interpreter for this service (here it is called `mapscript`). The interpreter links to a set of functions that perform relevant GIS functions. To enable the demonstration, we pass to the interpreter the name of the publishing script to be used (here it is in the file *circle.0*). The full call is therefore as follows:

```
http://lilliput.gov.lp/cgi-bin/gis/demos/mapscript?circle.0
```

The complete process needs two scripts. Firstly, a publishing script (`mapscript`) which defines initial locations for the two points involved and provides an entry into the *publishing service* shown in Figure 3.5. A second script *(circle.1)*, processes subsequent calls made from the form. The complete list of files involved is:

Downloaded By: 10.2.98.160 At: 08:01 23 Oct 2021; For: 9781482220162, chapter3, 10.1201/b19465-4



Value	Point A	Point B	
SELECT POINT	○	⊙	Submit
Latitude (e.g. -34.5)	-37.45	-33.53	Reset
Longitude (e.g. 134.5)	144.5	151.1	

FIGURE 3.4: A simple form interface for computing great circle distance.

- An initialisation script (*circle.0*);
- A processing script (*circle.1*);
- The document template in Figure 3.6, which is essentially the form shown in Figure 3.4 but with variables in place of the locations of the two points;
- The map used, saved as a static JPEG image (*great-circle.jpg*);
- The interpreter (`mapscript`).

The HTML file that is displayed on the client browser does not exist as a stored file. It is generated on the fly by the server, and the key element in the form is the image input type, given by the element:

```
<input type="image" name="coord" height="255"
                    width="320" src="great-circle.jpg">
```

Clicking on the image generates a pair of values, here called *coord.x* and *coord.y*, which hold the coordinate values of the point on the image where the mouse is clicked. These values are passed off to the server when the user selects the "Submit" button on the form.

The publishing script combines input from the form (Figure 3.4) with a template in Figure 3.6 by replacing actual values (e.g., 147) for the corresponding XML variables (`along`). The resulting HTML code of the new form is shown in the right column of Figure 3.6 and is passed to the processing service as XML.

Downloaded By: 10.2.98.160 At: 08:01 23 Oct 2021; For: 9781482220162, chapter3, 10.1201/b19465-4

source	circle.xml	Value	(define the template file)
target	STDOUT		(send results to standard output)
var	alat	-34.5	(initial Latitude of Point A)
var	along	147	(initial Longitude of Point A)
var	blat	-33	(initial Latitude of Point B)
var	blong	149	(initial Longitude of Point B)
var	radius	6306	(Earths radius in kilometres)
form			(extract fields from the form)
circle			(calculate the distance)
sub			(place new values in the template)

FIGURE 3.5: The publishing script *(circle.1)* used in the great circle example.

XML Template		**HTML Form**	
(input circle.xml)		(standard output)	
<input	type="text" name="alat" value="<alat>"	<input	type="text" name="alat" value="-34.5">
<input	type="text" name="blat" value="<blat>"	<input	type="text" name="blat" value="-33">
<input	type="text" name="along" value="<along>"	<input	type="text" name="along" value="147">
<input	type="text" name="blong" value="<blong>"	<input	type="text" name="blong" value="149">

FIGURE 3.6: Use of a document template to replicate the form used in the great circle example.

Figure 3.7 shows the HTML source code that creates the form used in the great circle example. There are four key parts to this HTML that are worth hightlighting. These are:

- `<form action="http://mapcalc.gov.lp/cgi-bin/gis/gc-calc.pl method="POST">` which is the call to the form processing script;
- `<input type="image" name="coord" height="225" width="320" src="great-circle.jpg">` listing an image input field, which ensures that coordinates are read off the map;
- `<input type="hidden" name="radius" value="6366.19">` representing a hidden field, providing a value for the earth's radius;
- `<td><input type="text" name="alat" value="-37.45">` `<td><input type="text" name="blat" value="-33.53">`

Downloaded By: 10.2.98.160 At: 08:01 23 Oct 2021; For: 9781482220162, chapter3, 10.1201/b19465-4

```
<html>
  <head>
    <title>Great circle distance calculations</title>
  </head>
  <body>
    <form action="http://mapcalc.gov.lp/cgi-bin/gis/gc-calc.pl
         method="POST">
      <table>
        <tr>
         <td><input type="image" name="coord" height="225" width
              ="320" src="great-circle.jpg">
          <td><h2>Great circle distances</h2>
           This service calculates the distance from the point A
                to point B.
           <br><i>Click on the image to select a new point.</i>
           <br>Lat/Longs are in degrees North.
           <br><b>Current distance
        <input type="text" name="circle" value="732.9" size
            ="10"> Km
        <input type="hidden" name="radius" value="6366.19">
      </b>
      </table>
    <p>
      <table border="1" cellpadding="1">
        <tr><td><b>Value</b> <td><b>Point A</b>
        <td><b>Point B</b>
        <td rowspan="2"><input type="submit">
        <tr><td>SELECT POINT
        <td><input type="radio" name="site" value="source">
        <td><input type="radio" name="site" value="target"
            checked>
        <tr><td>Latitude (e.g. -34.5)
        <td><input type="text" name="alat" value="-37.45">
        <td><input type="text" name="blat" value="-33.53">
        <td rowspan="2"><input type="reset">
        <tr><td>Longitude (e.g. 134.5)
        <td><input type="text" name="along" value="144.5">
        <td><input type="text" name="blong" value="151.1">
      </table>
    </form>
  </body>
</html>
```

FIGURE 3.7: HTML code which generates the great circle distance Web form

Downloaded By: 10.2.98.160 At: 08:01 23 Oct 2021; For: 9781482220162, chapter3, 10.1201/b19465-4

... <td><input type="text" name="along" value="144.5">
<td><input type="text" name="blong" value="151.1"> being the four lines of coordinate values processed by the publishing script.

This has been a simple implementation of a geographic query, where user actions on the client are processed by a server-based service. That is, mouse clicks are converted into coordinates and passed to server-based functions where they are processed. The server side code (not shown here) calculates the great circle distance and passes the result back to the client form where it is rendered.

With the increase in computing power on clients, including smartphones and tablets, geographic queries are now commonly carried out at the client, reducing the need for server based processing. These queries may include buffer searches and area calculations. More complicated calculations, such as processing vegetation statistics using large databases and answering spatial queries on proximity, would still be carried out on a server due to processing speeds and the amount of data involved.

3.8 Summary and Outlook

This chapter has introduced the basic concepts of server side processing:

- §3.1 introduced Web server technology, now part of our daily lives, and the HTTP protocol which underlies it. Some of the shortcomings of the first (and still widespread) version of HTTP were discussed.
- §3.2 introduced the software on the server side and §3.3 the sort of processing it carries out.
- Forms play a major role in much interactive Web processing and §3.4 gives a brief.
- Forms need scripts to process them on the server side, discussed in §3.5.
- The chapter ends with two examples of GIS processing, online map building in §3.6 and geographic queries in §3.7.

The examples in this chapter are indicative of the kind of scripting that developers of Online GIS are likely to encounter. Most commercial systems have in-built scripting features, which simplify the implementation of GIS online, much as we have demonstrated above.

Over the next few years there will be a significant transition from corporate owned data centres to cloud-based data centre infrastructure. As a result, there will be changes in the way server-side Online GIS operations will need to function.

There are already a number of cloud service providers that are offering cloud infrastructure that can dynamically expand or contract in terms of active server numbers. Running applications on these types of infrastructure requires significant ability for the server-side operations to also run in this ever changing *elastic* environment,

Downloaded By: 10.2.98.160 At: 08:01 23 Oct 2021; For: 9781482220162, chapter3, 10.1201/b19465-4

bringing its own set of challenges for setting up server-side components of Online GIS systems.

Downloaded By: 10.2.98.160 At: 08:01 23 Oct 2021; For: 9781482220162, chapter3, 10.1201/b19465-4