Part I

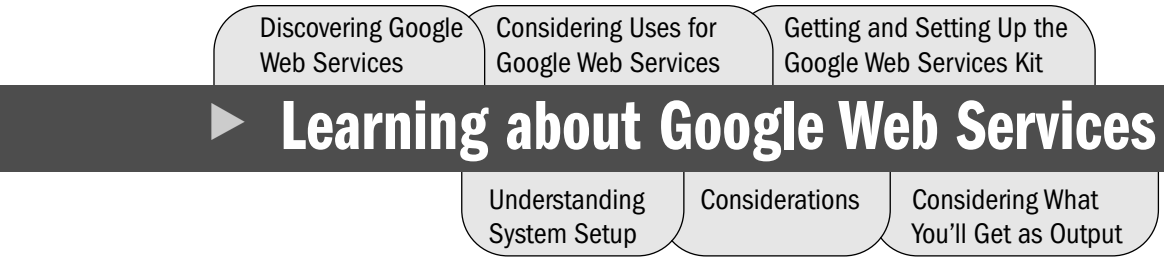
Discovering Google Web Services

1 Learning About Google Web Services

2 Defining a Search

3 Working with Web Service Data

Chapter 1



Google is one of the most popular search engines around because it provides a superior number of hits. Of course, more hits don't translate into better data. The search engine is also good at providing valid information through the use of indexing and filtering so long as you specify the search criteria clearly. Given the number of ways that the Google Advanced Search (`http://www.google.com/advanced_search`) helps you look for information, providing clear direction can be overwhelming to some. The flexibility provided by the interface is part of Google's charm, however, and the reason many power users prefer Google. If you can't describe a search using this interface, you might not know what you're looking for.

Google Web Services is a means of accessing Google without going to the Web site and performing a search manually. This Web service provides essential services by helping you automate the search process and presenting data in the form that you need, rather than in the form that Google thinks you need. In this chapter, you discover how Google Web Services can help you perform searches faster and with greater accuracy. The result is that you'll reduce the time you spend searching and actually have time to do something with what you find.

It's not too amazing that Google Web Services is extremely popular—you can use it to find information located on any public Web site. In many ways, Google Web Services is superior to the manual search technique because automating a search saves both time and money. Even companies such as Macromedia (`http://www.macromedia.com/support/flash/applications/google_search/`) are getting into the act by providing tutorials and other support for Google Web Services. Google Web Services lets you look for information in many ways. For example, you could use it as the search engine for a small Web site. This chapter also discusses how you can use Google Web Services in other ways, how to download and install the kit that Google provides, and what you should expect as output.

▶ NOTE

Various sources also refer to the Google Web Services as Google Web Application Programming Interfaces (APIs). The term *API* refers to a set of functions that a developer can call on to perform application tasks. For example, opening a file requires use of one or more functions provided by the operating system API. The term *Web service* is more specific and appropriate than API, so this book uses Google Web Services throughout. However, you can use the two terms interchangeably.

Understanding Google Web Services

Whenever a new technology appears on the scene, it's important to compare it with other technologies. The comparison process often helps you decide how this new technology differs from what you used in the past and reduces problems caused by hype. The media might try to convince you that a new product or service is something completely different, when in fact it's merely an update or a new implementation of an existing technology.

Currently, there's a lot of hype about Web services that makes them sound like something new and very complex. This section of the chapter defines Web services generally, examines Google Web Services specifically, and compares this technology to older technologies. What you'll find might surprise you because Web services are really a new implementation of an old technique.

▶ NOTE

Don't confuse *new* with *useful*. Web services are very useful because they add new functionality to an existing idea that has worked for a long time. They're also new in that they use a different process from other technologies. However, the technology itself builds on other techniques that you have already used in some way. In sum, the implementation is new, the process is useful, but the technique is the same one you've used in the past.

What Is a Web Service?

You can look at a Web service from a number of perspectives. The easiest way to view a Web service is as a means of obtaining access to information. Essentially, you ask the server for information and the server returns that information in some form. The request and the returned information normally appear in eXtensible Markup Language (XML) form. Using XML preserves the meaning behind the information, regardless of the diversity of the

platforms involved, so that you receive not only the information, but understand the context in which the information is used. The "Understanding XML Basics" section of Chapter 3 tells you more about XML. All you need to know now is that you receive information in XML format.

From a Google Web Services perspective, you request information based on any of a number of search criteria. Google supports a number of search techniques and not every technique works well for every kind of search. Chapter 2 discusses search techniques in detail. For now, just think of the search criteria as a form of request. The request defines the kind of information you want to know and how detailed that information will be. Google Web Services returns the information you request (when available) in a standardized format.

▶ NOTE

Google's database s*chema* specifies the format of the information. A schema defines the organization of information in a database. Fortunately, the format of the data returned by Google is relatively simple. You only have to consider a few return types. However, the content of the return data is a different story. Learn more about the Google database schema in the "Understanding the Google Data Output" section of Chapter 4.

A Web service also performs some type of useful work. The useful work might be something as simple as interpreting your request, calculating the answer, and sending the result back. In the case of Google Web Services, the Web service accepts your request in the form of a search request, interacts with the database through a search engine to obtain the information you requested, and sends the information back to you. The search can take various forms. For example, you don't have to search all Web sites—you can concentrate on just one. You might want to look for pictures, rather than text, and might only have an interest in newsgroups. The rest of the book shows how to perform all of these tasks. The main idea is that you can submit a variety of search request types—the request type affects the information you receive back from Google.

The final consideration for a Web service (at least from the Web service user perspective) is that it executes on the remote machine, not on your machine. In short, this means you're using resources on that other machine with the permission of the machine's owner. The remote machine can set requirements for using the Web service, as well as require you to perform specific setup and security checks as part of your request. In the case of Google Web Services, you need to obtain this permission by requesting a license. You also need to download the Google Web Services Kit to ensure you follow the terms of the licensing agreement. The "Downloading and Installing the Kit" section of this chapter tells how to obtain the required permission and what this permission means to you.

▶ TIP

You may find that Google Web Services is so indispensable that you'll want to work with Web services from other vendors. For example, Microsoft supports the MapPoint Web Service (`http://www.microsoft.com/mappoint/net/`). In time, standards organizations will set up directories of these Web services that you can access with ease. In the meantime, you can search for companies that offer Web services using the Web Services Finder page at `http://www.15seconds.com/WebService/`. Some people have problems using the Web Services Finder; it might produce an error instead of presenting a list of Web services. In some cases, you'll need to use a specialty Web service list such as the one at `http://www.flash-db.com/services/`. The Web services on this site are special because many of them perform one task well, such as providing you with a location based on a domain name.

How Do Web Services Work?

Many people fear new technology because they don't understand how it works, and many of those who do know how it works enjoy the mystique of knowledge too much to share it with anyone else. Web services are actually quite easy to understand if you look at them in a way that relates the task to everyday occurrences. For example, you might compare the operation of a Web service to making a withdrawal at the bank—the process really is the same. The one thing to remember is that the process a Web service uses to perform a task is always the same. No matter what technology you use to make a request or receive a response, the steps are still the same. Here are the steps that most Web services, including Google Web Services, use to complete a transaction.

1. *The client discovers the Web service*. During the act of discovery, the client might do things like download a file that tells how to interact with the Web service. This step is the same as someone walking into the bank. The person knows the bank exists and the bank teller might have noticed the person. The bank posts the rules for making a withdrawal or the teller might help a first-time customer understand the rules.
2. *The client makes a request based on the rules delivered during the discovery phase*. The rules might specify that the request has to appear in a certain form, and the client must provide specific data. This step is the same as the person walking up to the teller's window with a withdrawal request. The request must contain the person's account number, the amount they wish to withdraw, and other identifying information. The bank specifies the format of the request and the information it must contain.

3. *The server might ask the client for credentials depending on the openness of the Web service*. Google Web Services is public but still requires that you supply a developer license (account) number as identification. This step is the same as the bank teller asking you for a driver's license or other form of identification before honoring your withdrawal request.
4. *The Web service performs the work required to honor your request*. In most cases, the Web service accesses a database for information, it could enter an order, and it might even provide some level of formatting information about the original information (such as the typeface used for a word-processed document). Google Web Services performs a number of tasks depending on the request you make. The easiest request is a general search, but you can also perform checks such as making a spelling check. This step equates to the bank teller getting the money from the drawer and counting it.
5. *The Web service sends the data to the client*. The content of the information depends on the Web service. Google Web Services provides data in a very specific format based on the content of the associated database and the nature of the request. This step equates to the teller handing the person their money. In general, the teller orders the money in a specific way and counts it out to the person, rather than simply handing the money over.
6. *The client logs out of the Web service or the Web service disconnects the client after some period of inactivity*. This step equates to the person leaving the bank, money in hand. If the person doesn't leave the bank (they just hang out in the lobby), you can be sure that someone will ask them to leave.
7. *The client does something with the data it receives*. In many cases, it formats the data and presents it on screen for the user. This step equates to the person spending the money they receive from the bank.

You can add any amount of complexity needed to the individual steps, but these seven steps define the process every Web server follows. When you break a Web service down into these seven steps, the process that used to appear as magic suddenly becomes quite doable. Chapters 5 through 9 are essentially options you can use to perform these seven steps using different technologies. This book explores the seven steps using various languages and platforms—Google Web Services makes information available to just about anyone who needs it. However, it's important to remember that everything comes down to a client making a request and the Web server returning data.

Considering the Usage Requirements

There's no free lunch. Some people would have you believe that the Web service does everything for you and that the client does nothing at all. However, the client interacts with the Web service, which means the client must possess some intelligence to perform the task. To use a Web service, you must understand the usage requirements.

From a client perspective, the type of device you use to access the Web service determines the access speed, as well as what you can do with the data once you receive it. Although a PDA such as the Pocket PC can access Google Web Services just fine, you wouldn't want to use it to perform detailed searches or attempt complex activities such as converting data to another language. About the best you can hope for is to perform simple research. On the other hand, a desktop or laptop machine has all of the processing power, screen real estate, and functionality to perform any task. Google Web Services hasn't changed, but the capability of the client has.

▶ NOTE

This book discusses a number of mobile devices. The Pocket PC provides additional functionality and features that make it a better target for some types of applications than devices such as the Palm. On the other hand, most Palm devices are much easier to carry and cost less than the Pocket PC. This book examines the entire range of mobile devices to ensure you understand the limitations of using a specific device to access Google Web Services. I'm not saying one device is better than another—simply that one device works better than the other for a given application.

Google Web Services also has some usage requirements and these requirements might change the way that you use your client. For example, according to the license agreement (see Appendix B for details) you can't make more than 1,000 requests per day—at least, not without special permission. The request limitation ensures the Google servers won't become overloaded, but they also mean you must provide some type of monitoring in your application to prevent abuse of the licensing terms.

▶ WARNING

If you violate the licensing terms, Google Web Services simply denies your request. In addition, you might receive a message from Google requesting that you adhere to the terms of usage for the Web service.

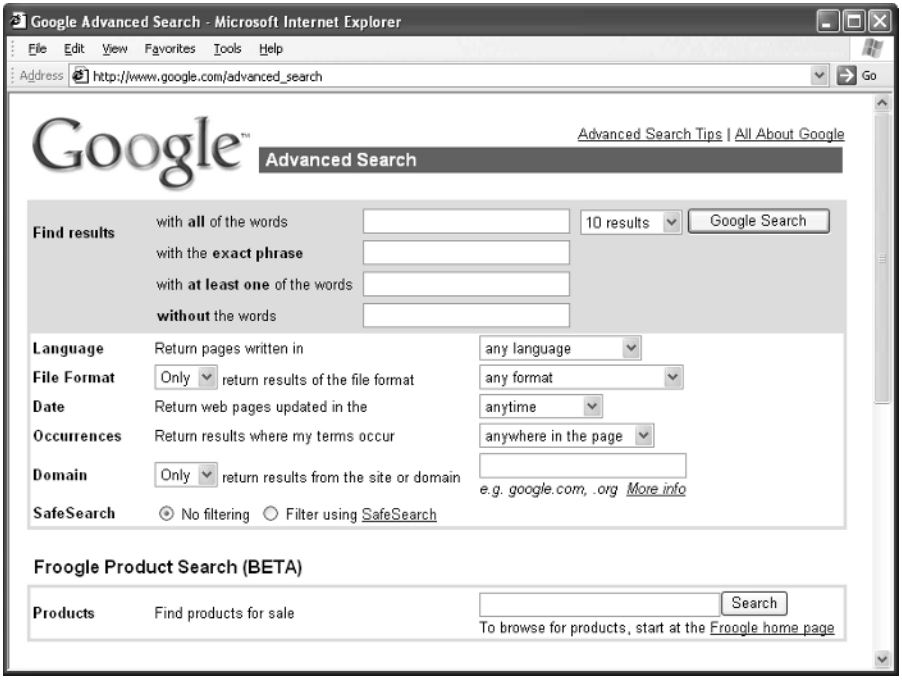
Often, you can get around the licensing requirements for a Web service by using smart programming techniques. For example, Google doesn't require that you refresh the information you receive at any specific interval. You determine when the information you receive is too old. Using good caching techniques means that you can create applications that are lightning fast, unless the request is new or the data is old. Although it seems as if a 1,000-request limit could cause problems, you can usually satisfy far more than 1,000 requests per day by using smart data caching.

Discovering Uses for Google Web Services

Everyone associates Google with searches of various kinds. Many people use Google for simple searches. In fact, you can set browsers such as Internet Explorer to go directly to Google whenever you enter a set of search terms in the address bar. One way to do this is to use a tool such as Tweak UI to create special search entries. You could also install the Google Toolbar (`http://toolbar.google.com/`), which has the option of making your default search engine Google. However, this book doesn't go into a detailed discussion of ways to manage simple manual Google searches.

This book helps you perform complex searches quickly, more reliably, and with less effort than any manual search can provide. Power users tend to use Google for intense searches, so they usually go directly to the Google Advanced Search page at `http://www.google.com/advanced_search`. (Some power users go so far as to memorize all of the special search terms Google uses so they can type everything in the basic search field.) Figure 1.1 shows an example of this page. Notice that you can manually search for a topic using a number of criteria, such as language and file format.

FIGURE 1.1:
Use the Google Advanced Search page to get a feel for the power of the Web service.



The Google Advanced Search page is useful because it helps you understand some of the power of the Google search engine. This page points to the need for some kind of automation in searching for information online. Not only does the page accept a number of inputs, but also permutations of the inputs will affect the output you see. Consequently, attempting to perform all searches manually is a time-consuming effort that many people would like to automate.

Now that you have a better idea of why you might not want to perform every search manually, it's time to consider specific ways to use Google Web Services. The following sections provide ideas on how you can improve productivity and make research easier using Google Web Services. The programming chapters of the book expand on many of these ideas by showing you how to implement them using code.

Performing Research

One of the most common uses of Google is performing research. Research searches normally begin based on keywords. The problem is that some keywords are ambiguous enough that the resulting data isn't meaningful. Try using *Windows* as a search term and you'll see this problem at an extreme. No one would want to look through all those hits. Using multiple keywords can help, but still doesn't solve the problem in many cases. For example, a search of the keywords *Visual Studio .NET Email* turns up 1,110,000 hits at the time of this writing—no one would want to go through that many hits looking for an application programming example.

Using additional search terms can help. For example, let's say that you're proficient in using C# and Visual Basic. Because many products use the term *Visual*, you could enter just *C#* and *Basic* in the "with at least one of the words" field of the Google Advanced Search page. However, you still end up with 276,000 hits—too many for the active developer to search.

Google Web Services can help in this case because you can automate multiple searches to locate specific information. For example, say you want to use a particular class or you have a special need in the application. Performing the search manually could require multiple trips to Google. However, using Web services means you could enter the criteria once and let the application make the multiple searches for you.

Even given the speed of an automatic search, you might wonder whether it's worth the effort of using Google Web Services. However, an application can do something that a manual search can't (at least not without a lot of trouble). Once the application returns from the search, it could store the results. The application would continue with each search scenario until it finished. Then the application could analyze the various returns and create a list of most likely sites based on the results. The 1,110,000 *Visual Studio .NET email* hits could suddenly become 20 or 25 hits that truly have useful information.

Conducting an Expansion Search

Expansion searches help you locate all available information on a topic by playing to the features that Google provides. For example, the order of search terms is important in the way that Google interprets a search. In addition, if you work in an acronym-laden field, expanding the acronyms is important to locate all sources of information on a given topic. Consider the following permutations of a search using the keywords *Visual Basic serial port*.

Visual Basic Serial Port This combination returns 132,000 hits with a first site of `http://www.distiworld.com/cd-burner-to-download.htm`.

Serial Port Visual Basic Just changing the two groups of words around reduces the number of hits to 130,000 with a first site of `http://www.lvr.com/spc.htm`.

Serial Port VB Using the *VB* acronym reduces the number of hits further to 58,200 with a first site of `http://www.control.com/1026175817/index_html`.

VB Serial Port You'd think that this number would be higher than the *Serial Port VB* search because of previous results. However, the number of hits is only 57,300 with a first site of `http://forums.basicmicro.net/ShowPost.aspx?PostID=7638`.

Four sets of keywords (and you could easily do more), four completely different results—it's not hard to understand why an expansion search could help you obtain the maximum benefit from Google. Manual expansion searches become cumbersome for a number of reasons. Repetition is one of the main causes, but there are others such as entry errors and result interpretation. You have to provide enough keywords to make a search specific, but each keyword adds an order of complexity to the expansion search.

Google Web Services steps in by letting you perform an expansion search automatically using code. You supply the four keywords—the code does the rest. By comparing the results of each expansion search, you can come up with an optimal group of sites. For example, you could verify that the site appears in every expansion search return, which tends to reduce the false positives. You can also rate the sites based on the number of times they appear and their position in the list. Although it's possible to perform this kind of data manipulation using a manual search, no one would want to do it.

Searching a Specific Site

Some Web sites don't provide a search engine. The site might be too small to support a search feature or hosted so the developer doesn't have access to the server's search feature. In other cases, a site does provide a search engine, but the search engine doesn't work nearly as well as Google's. You may find that the search engine fails to produce the desired results, even when you know the information exists. In both cases, you can create a site-specific search using Google Web Services.

You can perform this kind of search manually. In fact, it's not even all that time consuming. However, remembering the information you have to provide in an URL or going to Google's advanced search site every time you want to perform the search is a headache. Using Web services lets you store all of the static settings—the ones that won't change—so that all you need to know is what keywords you want to enter for that site. A site-specific search is all about convenience. Using this technique makes it easier to get the information you need without a lot of effort.

One way to use this technique is to create a search setup for your personal Web site. Many Web sites owned by individuals or the self-employed appear on hosted sites, making it impossible to add search capability with any ease. A Google Web Services application can make it easy to add a professional search service to your site, making it a lot more attractive to anyone who visits.

Another way to use this technique is to create custom search Web pages. I built one for my personal use that includes links to all my favorite coding sites. All I do now is select the site I want to search, add a few keywords, and Google Web Services takes care of all the hard work for me. Not only am I more productive, but I can stay focused on the task at hand—finding sample code. I can even make searches of multiple sites with a single click. Even though multiple searches take place in the background (a minimum of one search for each site), I only click the search button once.

Learning More about a Site

What do you really know about a Web site before you visit it? This question takes many people by surprise because they have to admit that they really don't know anything about the site. However, visiting a site implies that you're willing to open yourself to anything the site can provide within the limitations of your browser. A site that contains pornographic material or a virus when you're conducting legitimate research on parts of the human anatomy is an unwelcome surprise that you could avoid.

Google provides a number of searches you can use to verify the usefulness of a Web site before you visit it. For example, you can begin by looking for keywords in the snippet and site summary that Google provides. An examination of the links for the site, along with the Web information it provides is revealing. Figure 1.2 shows the results of an informational search on my Web site.

You can also conduct a related links search to see how the site connects to the rest of the Internet. (Chapter 2 discusses search types in detail, so don't worry if these specialized searches are unfamiliar.) If you're truly uncertain about the usefulness of the site, you can view a cached version of the page. The cached version does contain old data, but it can help you check for objectionable terms or content without exposing yourself to as much risk. The point is that with the security problems that users face today, they need a better way to assess

the risk of visiting a particular site online. A fact-finding search is very useful in keeping some types of Internet risks at bay.

Unfortunately, few users are going to take the time to perform such fact-finding before visiting a site. It's simply easier to click on the URL and go there. However, you could build a Google Web Services application that would display the search results and assess the potential of a Web site before the user visits there, while maintaining single click efficiency. When the user clicks a link, your application can check the site in the background and verify that it's reasonably safe. What the user sees is the normal sequence of events that take place when they click the link.

Getting Old Data from the Cache

The Internet is constantly changing. In fact, it changes so fast sometimes that it's hard to keep all of the links updated. Anyone who spends any amount of time researching information online knows that even the links Google provides get outdated. However, seeing an error message, page not found, when you click that link isn't the end of the road. You can request cached data from Google. The cached information is old in many cases, but at least it's available and you can use it for whatever you need. Figure 1.3 shows a typical example of a cached data page.

Like many other kinds of Google searches, you can perform a cache search manually. However, you have to perform multiple keystrokes to perform the search, assuming you remember to do it. Many people simply move on to the next site without thinking when they reach an error message.

FIGURE 1.2: Informational searches help you learn more about a Web site before you visit.

FIGURE 1.3:
Cached data searches can be very helpful, especially during research.

A Google Web Services application can reduce the problems of the dead link. It could begin by searching for the site. If the site isn't available, the application can move on to the cached page. When Google doesn't provide a cached page (a rarity), the application can move on to related links. Even if these techniques fail, the application could use some kinds of regressive searching. A regression search is one in which you begin with the result data and look for the information used to create the results. The point is the user wouldn't see an error message—a page of some kind would display and the user would then make the decision on the value of that page.

Performing Spell Checking

Interestingly, the spelling check is one of the few Google Web Services tasks that you can't perform manually. To use this feature, you send a string (up to 2,048 characters long) to Google Web Services. The Web service checks the string for spelling errors and sends the corrected string back to you.

At first, you might wonder how you would use this service. After all, it's relatively easy to find a local spell checker that won't use one or more of the 1,000 calls that Google allots to each developer per day. The answer is that you wouldn't use this service personally in most

cases. However, if you're running a Web site that requests text input from users, you can use the spell checker to validate their work.

Because the data you receive as input from the user contains fewer errors, you'll also end up doing less work. For example, any database you use to maintain the user input will have fewer errors, so you'll spend less time looking for errant records.

Avoiding Pornographic Material

The Internet contains all kinds of pornographic material. No matter what your personal preferences are, this material becomes annoying at some point because it tends to get in the way of legitimate research. In addition, you don't want children to see this kind of material, and it can cause problems in the workplace. Fortunately, Google does provide a means of searching the Internet without running into too much pornographic material. In fact, you can theoretically eliminate all of it through wise keyword search choices.

Google provides an actual search feature that blocks pornographic sites based on your choice of keywords. The feature does work for the most part, unless your selection of keywords is less than perfect. For example, using *breast* as one of the keywords in a safe search produces a number of sites for cancer research and many forms of help or assistance. Using the standard search produces the expected results (13,000,000 of them). Unfortunately, figuring out which keywords to avoid isn't always easy.

Like many of the other tasks discussed in this section, you could perform this task manually and might even get good at it given enough time. However, Google Web Services can make the search process a lot more efficient. For example, you can create an application to perform a keyword translation to help you avoid the terms that produce pornographic results. When you couple this application with the safe search feature, all you'll receive is sites that contain the kind of information you need.

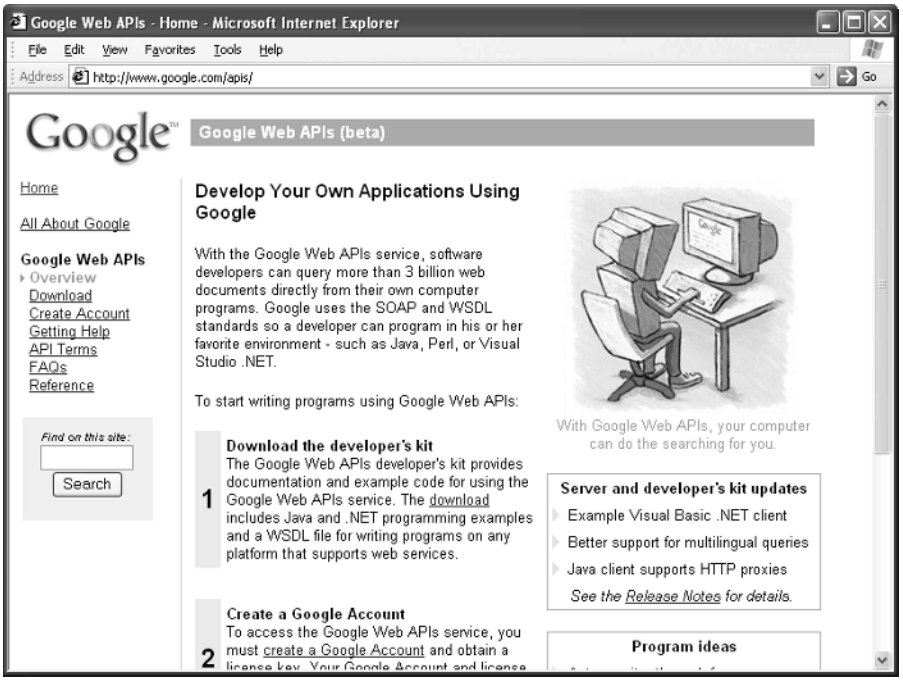
Downloading and Installing the Kit

Before you begin working with Google Web Services, you need to obtain the kit and a developer license. Once you have the kit, you need to install it and become familiar with its content. The following sections describe the kit-related tasks you need to perform.

Performing the Download

Downloading the kit is easy. You'll find the main Web services page at `http://www.google.com/apis/`. Figure 1.4 shows that this page contains information, along with two important links. Although the steps shown in the figure are numbered, you can perform the first two steps in any order. This chapter assumes that you want to download the Google Web Services Kit first.

FIGURE 1.4:
You can obtain both the kit and the developer license on this site.



Click the download link and you'll see a page that describes the kit. This page also includes the licensing agreement for the Google Web Services Kit. Make sure you read the licensing agreement and understand what it means before you proceed (Appendix B provides a licensing agreement checklist you can use for your applications). Don't worry about copying the licensing agreement to disk—the kit includes a copy of the licensing agreement you can use for reference purposes later. Check the "I have read and agree with the Google Web APIs license terms" option and then click Download Now. You'll see a File Download dialog box. Click Save and you'll see a Save As dialog box. The default name of the file is `GoogleAPI.ZIP`, but you can save it using another name if desired. Click Save and you'll receive the file—a mere 666 KB in size.

Getting a License

Once you complete the download process, click the Create a Google Account link that is available on the same page as the Google Web Services Kit download. You'll see a Web page that requests an email address and password. This page also contains links to Google's terms of service and privacy statements. Make sure you read both before you proceed. The Google Web Services Kit doesn't include copies of either document, so you might want to copy the

information and save it on disk for use later. When you finish reading both documents, check the "I have read and accepted the Google Terms of Service above and Privacy Policy" option and click Create My Google Account.

▶ WARNING

The process described in this section doesn't always work as anticipated. In some cases, Google displays an error message during the email verification process. In other cases, you may think the email verification worked correctly, but never receive a confirmation email from Google containing your license key. When either of these problems occur, contact Google support at `accounts-support@google.com` for assistance. The support staff will usually send another confirmation email to your inbox that you can use to confirm your account. Never assume the process has worked until you receive the license key.

At this point, you'll see a message stating that Google will send a verification message to your email. Click on the link provided by the verification message to activate your account. After you verify your account, Google will send your developer license to your email. The license normally arrives in about an hour—you might need to wait more or less time depending on how busy Google is at the moment. Make sure you save the email message containing the developer license because you'll need it for every transaction later.

▶ TIP

You can always change the password and other information associated with your account. Simply go to the Google Accounts site at `https://www.google.com/accounts/Login`. Type your name and password to enter the site. Select the My Account link to change the account information.

Installing the Kit

The kit is actually a Zip file containing examples and documentation. If you're running Windows XP, the operating system provides a program to unpack the file for you. Otherwise, you'll need a special program that reads the compressed file and unpacks it for you such as WinZip (`http://www.winzip.com`).

You won't find any actual developer tools in the Zip file. The file does include complete path information, so you can unpack it in the root folder of your hard drive if you like. I used the D drive on my system, so the Google Web Services Kit appears in the `D:\GoogleAPI` folder.

At this point, the kit is ready for use. However, before you go any further, you need to know about two files in the \GoogleAPI folder. The LICENSE.TXT file contains a copy of the license agreement that you saw online. Make sure you retain this file so that you can refer back to the usage terms as needed.

▶ NOTE

Google will eventually update their Web services package and could change the licensing agreement as well. You might wonder whether the online version of the agreement overrides the version of the agreement that comes with the Google Web Services Kit that you downloaded. Unfortunately, Google doesn't address this concern in their license and not being a lawyer, I can't advise you. If you have questions about the terms of using Google Web Services, make sure you contact Google at api-support@google.com.

The README.TXT file contains useful information about the Google Web Services Kit and tells you where you can obtain additional information. This file is very helpful because it contains URLs where you can obtain additional examples. It also has URLs for help sites and additional information. Finally, you'll want to read this file if you want to run the examples because it contains instructions for using them. Interestingly enough, even though the kit doesn't include a Practical Extraction and Reporting Language (PERL) example, this file also includes instructions for using Google Web Services with PERL.

System Setup Considerations

Once you obtain the Google Web Services Kit and a developer license, it's easy to think that you're ready to write your first program. Theoretically, you can do just that. The problem with proceeding at this point though is that you don't know about the viability of your system configuration. For example, if you have a very fast processor and a lot of memory, it's easy to assume the page you've designed will work fine on all systems. However, once you load the resulting application on someone else's machine, it might not work very quickly (if at all).

Defining a usable development setup can save you considerable time and effort later. When you create a great development environment, you ensure that you'll see the application as the user does, which reduces the potential for deadly errors. Because the Google Web Services Kit is so accommodating, you'll need to spend a little extra time considering all of the possible usage scenarios. The following sections provide tips you can use to reduce the setup complexity.

Understanding Connectivity Requirements

You must consider three kinds of connectivity when you set up your development system. The first level of connectivity is your own machine. Make sure your machine has a connection to the Internet. Otherwise, any tests you run will fail. Remember that a Web service runs on the remote machine, not your local machine. You're borrowing the resources of that remote machine to perform useful work.

The second level of connectivity is the user's machine. If you create a Web site that simply contains links to Google's Web site, you can assume the user has a connection, but how fast is that connection? The best Web sites I've seen ask about the user's connection speed. This question allows the application to send the user the level of information that their connection can comfortably support. If you know that most users will rely on a dial-up connection for your Web site, make sure you also use a dial-up connection for testing. This additional step can greatly reduce the chances that you'll make the application too robust. Users who leave your site and don't use your application are users who are probably visiting someone else.

The third level of connectivity is the non-connected mode. You need to consider what happens when the user loses the connection or doesn't have one available. Applications can store static data locally to enable the user to continue using data they have already queried from the Web service. However, you need to observe any refresh requirements and ensure the data retains the same information the user would see online. For example, the local copy of the data must include any required copyright statements or trademarks.

▶ NOTE

Google's licensing terms are flexible in that they allow you to store information as long as that information remains viable to you and you retain your relationship with Google. This flexibility means you can create user applications that only query Google when necessary, instead of for each request. It's important to note that any application you create using Google Web Services will require your license to access the site. Any queries a user makes using your application will count against your licensed access total for the day. The best policy is to ensure the user obtains a personal license from Google whenever possible. Your application can request this license information from the user so the user's access doesn't count against your total.

Programming Setups for the Non-Developer

Many of the people reading this book have marginal experience with programming or do it as a hobby. It's true that Web services rely on the resources of the remote machine, but it's also true that the client must perform work too. If you have a machine that's already marginal—

that doesn't run applications well—trying to write a Web service application for it could make matters worse. The local machine must have resources for using the Web service application.

▶ NOTE

This book doesn't teach you how to program, so make sure you spend at least a little time learning one of the programming languages discussed in this book before you begin working with the examples. I do provide good descriptions of the applications, but these descriptions won't be enough if you don't understand basic programming concepts.

Depending on the kind of application you create, you'll also need local resources for the programming environment. For example, VBA users have not only the Office application of choice running, but also the VBA development environment. The addition of the VBA development environment can reduce your system performance to a crawl and give you unrealistic performance for your application.

It's also possible for you to speed things up too much. If the target platform is a 400MHz Pentium and you're using a 3GHz development machine, your application performance will look nothing like the user's performance in most cases. For a Web site, the machine performance differences might not be quite as significant as when you develop applications that run on the desktop.

Considering the User

Depending on how you use the Web application you build, user needs will take on significant importance. Many applications start out as projects that the developer is creating for personal use. Some of the best applications I've written fall into this category. However, taking shortcuts in developing the user interface, even if you're the only user, is never a good idea. At one time, I wrote rough applications that I understood but couldn't use efficiently because they were only for test purposes. After I ended up rewriting a number of the applications because I couldn't figure them out or other people asked me for copies, I began writing every program as if it were for someone else.

The applications you write with Google Web Services will likely see use from other people, even if you don't know it right now. Consequently, you need to consider what a hypothetical user will need. For example, you might need to include a few special search options. Sure, you could get the same results by typing a little extra text, but adding the functionality directly into your application makes it easier to use (faster in most cases as well).

It's also important to consider users with special needs. The "Addressing Users with Special Needs" section of Chapter 11 contains details on this topic, but you might need to perform setups before you even begin coding. For example, if you work on a Windows machine, you'll probably want to set up the Accessibility features (these features normally appear in the Control Panel and within the `Start\Programs\Accessories\Accessibility` folder).

Using Multiple Test Devices

If your application will appear on the Internet, you need to test using multiple devices. It's no longer safe to assume that only desktop users will have an interest in your application. You might attract Personal Digital Assistant (PDA) and cellular telephone users as well. This is especially true of a Web application that helps users find a particular kind of information quickly. People often rely on these applications when time is tight and they don't have time to look for a product themselves.

▶ NOTE

Not every developer is concerned about writing applications for every platform—sometimes it's a matter of time; other times it's a matter of skill or perceived need. When an application you write falls into this category, you can still provide a modicum of support for wireless users by directing them to Google Wireless Services at `http://www.google.com/options/wireless.html`.

It would be nice if everyone could afford to test every application on every device, but that's not realistic for the developer. Sometimes you need to use an emulator to perform the testing because you don't have the real device handy. Fortunately, you can find a vast array of useful emulators on the Internet—everything from the Pocket PC to cellular telephones of all types. Emulators have limitations, but they do make good test devices in many cases. We'll discuss the advantages and concerns of using emulators in the "Working with Emulators" section of Chapter 9.

Sometimes it also helps to have multiple desktop machine setups. For example, you might need to consider how a Web page looks and acts in Netscape versus Internet Explorer. (Theoretically, you can run both browsers from the same machine, but doing so causes interference problems that some developers find distasteful.) Differences in how the browsers react to specific Web page designs could cause problems in your application. In some cases, you'll need multiple machines to perform this kind of testing. For example, you might need to consider how the application looks on a Macintosh versus a PC if your application has broad

enough appeal. Obviously, you can still write Google Web Services applications if you don't have a multiple machine setup, but having more than one machine does make development tasks a lot easier and less error prone.

Emulating the Real World

Developers often live in a laboratory. In the laboratory, everyone has the proper equipment, fast machines, and an even faster connection. The user never disconnects unexpectedly and always knows how to get the most out of their computer. The problem with the lab is that it doesn't model the real world. In the real world, users get bored, try odd key combinations just to see what they do, don't understand their computer very well, but do know how to complain about the smallest application problems. If you want to avoid problems with the application you develop, you need to create a development environment that models the real world.

It's also easy to get lost in the development environment setup. Make sure you understand the person who uses your application. For example, it's quite possible that only desktop users will have any interest in your site on desktop machine maintenance, but you need to determine that fact in some way (online surveys work well). You also don't want to spend a lot of time testing the application to meet the needs of users who have no use for your product. Again, surveys and newsgroup polls are helpful in determining the real world environment that you must emulate with your system.

Knowing What to Expect as Output

For many developers, the idea of a Web service is easy to grasp—knowing what to expect from it is hard. The output begins with a certain amount of raw data that you'll receive from the Web service. However, the raw data doesn't really define the Web service output completely. You also need to consider quantifiable components such as the input to the Web service and that data manipulation you'll perform. In addition, there are variant elements to the output, such as the timeliness of the data. Finally, you need to consider the intangible elements. The output has some value to you, but someone else will view the output in another way. Concepts such as relevancy are difficult to quantify or even define.

Google Web Services is no different from any other Web service when it comes to output. You'll provide input, receive raw data, manipulate that data in some way, and view the output—the result of everything you have done with the Web service. The following sections discuss various elements of Web service output as they relate to Google Web Services. These sections provide an overview—the book continues to explore the subject in other chapters. However, this is the starting point—the point at which you start to consider what to expect as output from your efforts.

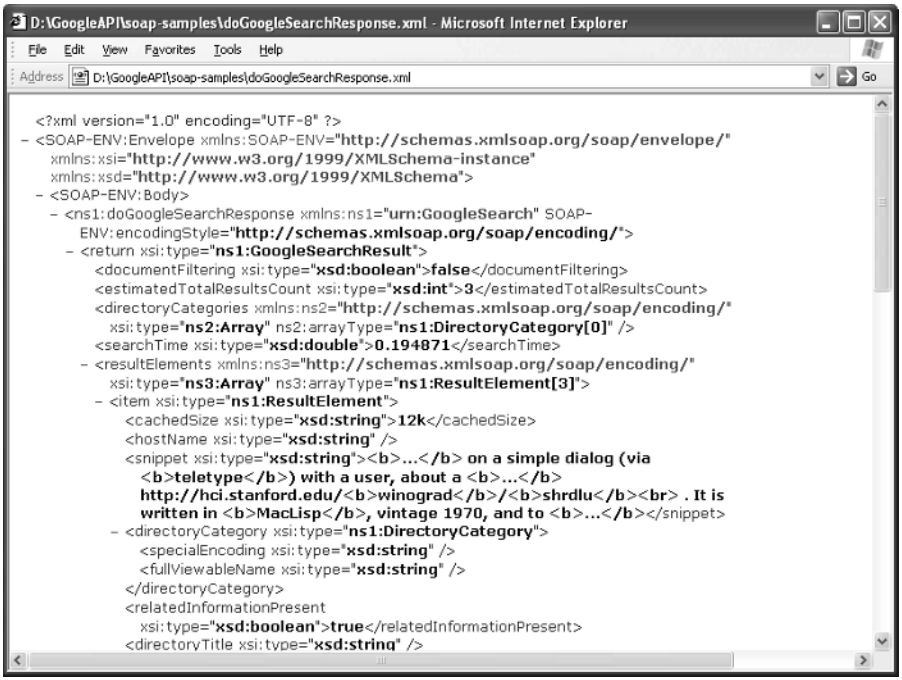
Limitations of Google Web Services Output

Many developers are used to working with a variety of data types when creating applications. Data types help define the kind of data you're using. For example, if a data element is a number, you might use an integer (a number without a decimal) or real number (one that has a decimal and equates to a Single or Double for Visual Basic developers). A Web service has no concept of data type when it comes to the data itself. Every data transfer is text. The XML used to transfer the data does include type information, but of the sort that's normally associated with database fields, which means you have to know the field names to make an interpretation. For example, you might receive data in a message like the one shown here.

```
<item xsi:type="ns1:ResultElement">
   <cachedSize xsi:type="xsd:string">12k</cachedSize>
   <hostName xsi:type="xsd:string" />
   <snippet xsi:type="xsd:string">
      <b>...</b> some text <b>highlight</b>) more text <b>...</b>
   </snippet>
   <directoryCategory xsi:type="ns1:DirectoryCategory">
      <specialEncoding xsi:type="xsd:string" />
      <fullViewableName xsi:type="xsd:string" />
   </directoryCategory>
   <relatedInformationPresent xsi:type="xsd:boolean">
      True
   </relatedInformationPresent>
   <directoryTitle xsi:type="xsd:string" />
   <summary xsi:type="xsd:string" />
   <URL xsi:type="xsd:string">
      http://www.mwt.net/~jmueller
   </URL>
   <title xsi:type="xsd:string"><b>DataCon Services</b></title>
</item>
```

You don't have to understand the XML portion of this message segment, but look at the data. Google Web Services sends all data as characters (as do all other Web services) and defines the data using tags (the words between the angle brackets) and attributes (extra information within the tag). For example, the line that contains `<URL xsi:type="xsd:string">http://www.mwt.net/~jmueller</URL>` includes the `<URL>` tag that tells you that this value is `http://www.mwt.net/~jmueller` and that the tag type is an `xsi:type="xsd:string"`. The tag tells you what kind of information this is. By knowing the Google database layout, you also know the data type and other information about the entry. However, the information you receive from Google is still plain text. You can see other examples of XML responses in the `\GoogleAPI\soap-samples` of the kit. Simply open them using Internet Explorer or another browser that supports XML. Figure 1.5 shows a typical view of one of the examples in the folder.

FIGURE 1.5: View example files using Internet Explorer or other XML-compatible browser.



Your browser is actually very handy for viewing XML data, even if it might not make sense right now. The "Viewing XML Data in Your Browser" section of Chapter 3 discusses in detail how you can use your browser. For right now, all you need to know is that you can look at the various kinds of XML responses by opening the files in your browser.

Figure 1.5 points to another potential problem with Web service output. All of the tags and other information supplied in a request and response consume space. The file is larger than a text file with the same data because of all the tag information required. In addition, it's far more efficient to store many data types in their native format, rather than use characters. Consequently, Web service data suffers from bloat. The data uses more bandwidth than a binary message and consequently, you could experience performance problems. Because of this issue, you need to create efficient queries for your application that maximize data throughput despite the limitations of the XML format. The "Making Sensible Queries" section of the chapter discusses this issue in detail.

The results you obtain from Google are largely a matter of the input you provide in the form of a request. The "Conducting an Expansion Search" section of the chapter points out a serious flaw in making any assumptions about the return you receive from Google. The query can become quite complex because even the order of the words makes a difference in

the results you receive. Google must make this assumption because most people enter the words in the order they think about them, which is usually most important to least important. Consequently, if you always assume that your first query returns all possible results, you'll find Google Web Services disappointing.

The ranking of results you receive from Google Web Services is also unlikely to be the same as the ranking you need. Google sells keywords to make some sites turn up higher in the result list. In addition, Google often bases the site ranking on criteria that won't match your own, such as the number of times that a keyword appears. The bottom line is that the output you receive from Google is "raw" output—information that you haven't filtered or organized in any way. One of the reasons to use Google Web Services is to enable you to perform tasks such as site ranking so the results appear in the order that's best for your organization.

Making Sensible Queries

Google Web Services can help you perform a number of tasks. The problem is that each request and response consumes resources. To get the most from this Web service, you need to optimize the requests and responses so that the value of the information you receive exceeds the cost of transmitting and manipulating the data.

Creating a request and then handling the response has several costs associated with it. Some of the costs are real world in that you must provide the infrastructure required to perform the task. Inefficient queries could mean adding additional bandwidth capacity or providing additional servers (if you make enough queries). Some costs are employee related—inefficient queries mean more waiting time as the computer crunches the data. Finally, inefficient queries can incur intangible costs. For example, people can become frustrated with poor query results, which affects their performance. Some of these costs are impossible to measure accurately, but they're real.

I often rely on the online search engine to help tune queries. Using the Advanced Search (`http://www.google.com/advanced_search`) page shown in Figure 1.1 can help you define and tune searches to obtain maximum data with minimal resource use. For example, computer technology is quickly outdated, so I normally provide a date range as part of my search. Using the online search to customize the date range for specific keywords can greatly enhance performance.

The Advanced Search page can help you tune keyword order—making it possible to reduce the number of keyword permutations you use on an expanded search (see the "Conducting an Expansion Search" section of the chapter for details). It also helps you decide on how to use permanent keywords. For example, a site that sells a specific product might include the product name as a permanent search term—one that is always included even if the user doesn't specify it.

One of the features the kit provides is a more complete list of special phrases and characters you can use for a search. Although the Advanced Search page will help you ferret out many of these search features, you won't find them all. For example, the Advanced Search page includes a blank for a search phrase, which is different from a keyword in that the search phrase must appear as specified on the target page (keywords can appear in any order). Fortunately, all of the special keywords and characters mentioned in the kit also work on the Advanced Search page so you can try them out. (Chapter 2 discusses search techniques in detail.) Depending on which special features you use for a search, Google Web Services output might not provide the information you imagined. Consequently, it pays to try these special features out to see what effect they have on your search results.

▶ TIP

You might wonder why I'm suggesting such heavy use of the Advanced Search page. Google allows you to make 1,000 requests per day using Google Web Services. The Advanced Search page doesn't have such a limit—making it easier to keep testing search techniques until you find the technique you want to use in your code. At that point, you can start making requests from Google Web Services to test your code. Don't waste calls on search techniques.

Even when you create a perfect search and properly filter the results using code, the information you receive from Google Web Services might not fulfill every need. At some point, you need to perform some level of human filtering. Users will need to state a preference or define how well a particular search result works. Only by tuning the filter can you hope to obtain specific results from Google Web Services. Tuning makes it possible to reduce search times from hours to minutes.

Defining Static and Dynamic Data

Web applications can include the concept of static and dynamic data. Dynamic data is the best type to use for Web services because it reflects changes in the Google database. An application gains important benefits by using dynamic data. For example, you won't try to access an old Web site that Google used to list because the dynamic nature of your application automatically removes the link from the list of results.

Unfortunately, dynamic data can also cause problems. For one thing, you need a connection to the Internet to work with dynamic data. When you use a desktop machine, maintaining a connection usually isn't a problem. However, many third party developers are working on applications where a connection might not be available, such as a research list application

for a PDA. You download the information from Google Web Services and then use it to create a report while on the road—the connection doesn't exist while you're on the road so the data is no longer dynamic.

▶ NOTE

Google does support the concept of cached data that is stored from the original Web site, but even the cached data ages at some point and becomes unavailable. Cached data does have an important use. For example, you can use cached data to obtain copies of old articles that a Web site no longer carries.

Using the term *dynamic* to refer to application data is also somewhat of a misnomer. Nothing is truly a dynamic data application. The moment the response to your query leaves the Google server, it begins to age. The data doesn't change once it leaves the Google server, so in reality it isn't truly dynamic. The only way you can achieve a dynamic presentation of sorts is to make multiple queries. You must define how often is often enough for your needs. Google doesn't provide any guidance in this case because search result viability varies by person, focus, and need.

These facts lead into the discussion of static data. Truly static data never changes at all. Most Web sites still rely on static data presentation because the information they display doesn't change often enough to warrant a dynamic presentation. When you make a single query to Google Web Services, the response you receive is static data. It's a snapshot of that particular part of the database at a specific time. The data won't change unless you make another query.

Understanding the static and dynamic nature of data is important when you design an application that relies on Google Web Services. Errors creep into the presentation you create as the data from Google Web Services ages on your system. Part of the design process for your application is to determine how much error you can accept.

Your Call to Action

If you've read the entire chapter, you know what a Web service is, how the Google Web Services fits within the general definition of a Web service, and what you can use the Google Web Services to do. You can use this knowledge to create opportunities to exercise Google as a search engine for all kinds of tasks. Data mining is increasing in importance as companies

strive to gain more from the resources of the Internet—this technique accesses the needed information and discards unneeded information. At this point, you also have a machine that's setup to create a Google Web Services application of some sort and you have the Google Web Services Kit installed.

The next step of the process is to evaluate where you're going based on the content of this chapter. You need to consider what you want to do with the information Google provides, how you plan to present it, your own capabilities, and the capabilities of the person using your application. This may sound like a lot of work, but it's important to create a firm foundation for your application. When you take these preliminary steps, you begin thinking about problems and solutions to those problems.

Chapter 2 builds on the knowledge you gained in this chapter. The emphasis of Chapter 2 is on data mining—the process of using specialized search techniques to whittle search results down to just the links you need. In many cases, you can also access these search techniques using Google search page, but the emphasis of data mining is automation. Only by using the Google Web Services can you automate the search process and then display the results in the form you want, rather than rely on Google's formatting methodology.