

Dr. Bob's prescriptions N°6: Searching Google (again) using Delphi 6 and Web Services

by Bob Swart

Last time, I showed you how to use the GoogleSearch using C++Builder 6 and Web Services. However, between the time of writing and the time of publication, something has happened. To cut a long story short: the GoogleSearch web service itself has been taken offline. It seems that only Google itself has the right to publish a web service to search Google, and that's exactly what they did (as well as to ask the competition to disconnect their Google search services).

So, this time, I'll examine the official Google Search Web Service, and show how we can use it with Delphi 6.

Google Web APIs

Information about the official Google Web APIs (beta 2) - as they call them - can be obtained from the Google website at <http://www.google.com/apis/>. I don't know why they call them simple APIs while in fact they are APIs made available as a web service. Anyway, at the aforementioned URL, you can see that it takes a mere three steps to start to use the Google APIs:

1. Download the developer's kit
2. Create a Google Account
3. Write your program using your license key

The first step is easy, and consists of downloading a 658,031 Byte ZIP file with the complete Google API (currently at Beta 2, released April 11th, 2002), samples for Java as well as .NET, an API reference and - most importantly for us - the GoogleSearch.wsdl file which contains the WSDL (Web Service Description Language) definition for the GoogleSearch web service. We'll use this file as starting point in step 3, when we're building our web service client with Delphi 6.

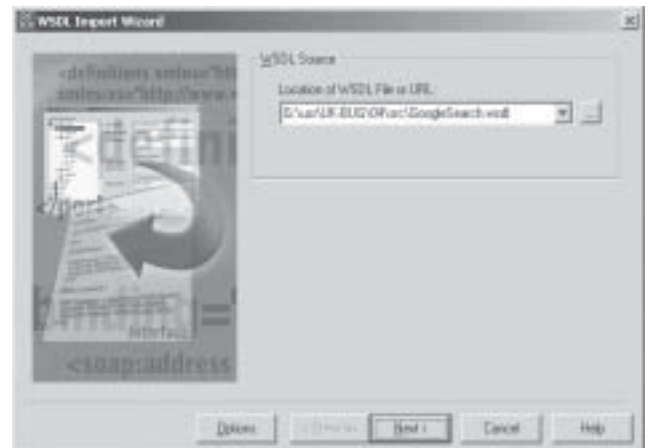
Google Search Key

The second step involves creating a Google Account. This sounds more dangerous than it really is. The use of the Google API is free (at least for now - or let's say: at least at the time of writing), but you need to pass a personal key that will allow you up to 1,000 search queries per day for non-commercial use only, etc. In order to get this key, you need to register yourself with an existing e-mail address such as ukbug@eBob42.com and a password (although you can quickly forget that password). An e-mail message will be sent to the specified account in order to verify the e-mail address. After you've received the e-mail message and clicked on the link inside, you will receive a second message with your special Google Search Key. In my case, that key is 1WpiIaxr+k+hbyYbRLZOJfg7X9Ngl837. The key is included in the source code, so the project executable will work right from the start, although it will only work

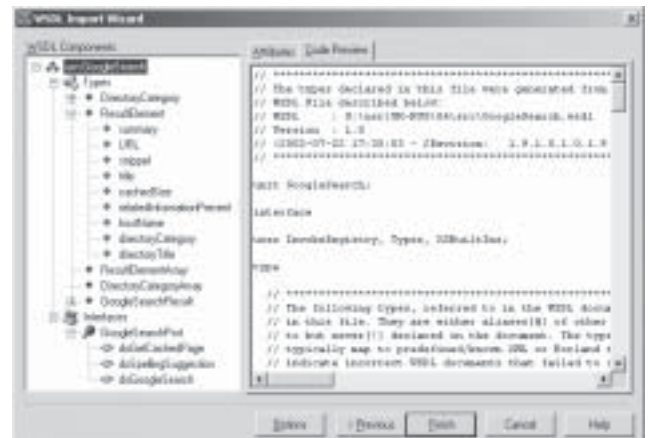
1,000 times each day (for all combined users of the executable), so you may want to register yourself and get your own personal Google Search key which entitles you to 1,000 search queries for yourself.

Google Search

Armed with the GoogleSearch.wsdl and the Google Search Key, we can start Delphi 6 (Professional or Enterprise) and build our web service client. Start a new Delphi project and save the (empty) main form in MainForm.pas and the project itself in Google42.dpr (like last time). The first thing we need to do now is to generate a Delphi import unit for the GoogleSearch.wsdl definition, so do *File / New - Other*, and select the WSDL Importer wizard from the Object Repository. Instead of specifying a URL for the Google Search, you can use the local GoogleSearch.wsdl file - this is especially handy in case you do not have a live internet connection available at all times, because all information is in the local WSDL file.



Click on the Next button, which will show a preview of the generated import unit as well as a treeview with the GoogleSearch specific types, interfaces, and methods.



As we can see in the treeview, there are three types: DirectoryCategory, ResultElement, and GoogleSearchResult, and one interface called GoogleSearchPort with three member functions: doGetCachedPage, doSpellingSuggestion, and doGoogleSearch. To illustrate the things we can do with the result, I've listed the types with their published properties below:

```
type
DirectoryCategory = class(TRemotable)
published
    property fullViewableName: String;
    property specialEncoding: String;
end;

ResultElement = class(TRemotable)
published
    property summary: String;
    property URL: String;
    property snippet: String;
    property title: String;
    property cachedSize: String;
    property relatedInformationPresent:
        Boolean;
    property hostName: String;
    property directoryCategory:
        DirectoryCategory;
    property directoryTitle: String;
end;

ResultElementArray = array of ResultElement;
DirectoryCategoryArray = array of
    DirectoryCategory;

GoogleSearchResult = class(TRemotable)
published
    property documentFiltering: Boolean;
    property searchComments: String;
    property estimatedTotalResultsCount:
        Integer;
    property estimateIsExact: Boolean;
    property resultElements:
        ResultElementArray;
    property searchQuery: String;
    property startIndex: Integer;
    property endIndex: Integer;
    property searchTips: String;
    property directoryCategories:
        DirectoryCategoryArray;
    property searchTime: Double;
end;

GoogleSearchPort = interface(IInvokable)
['{0B396A82-A4DD-69A7-A771-6D80F8831A71}']
function doGetCachedPage(const key:
    String; const url: String):
    TByteDynArray; stdcall;
function doSpellingSuggestion(const key:
    String; const phrase: String):
    String; stdcall;
function doGoogleSearch(const key: String;
    const q: String; const start: Integer;
    const maxResults: Integer; const filter:
    Boolean; const restrict: String; const
    safeSearch: Boolean; const lr: String;
    const ie: String; const oe: String):
    GoogleSearchResult; stdcall;
end;
```

The DirectoryCategory is a type that I won't use, but the ResultItem and GoogleSearchResult are the two result classes that will be used in this article. The GoogleSearchResult contains the resultElements property that points to an array of ResultItems. Finally, the function goGoogleSearch of the GoogleSearchPort interface is the most interesting (and the topic of this article), so let's examine that one in more detail.

doGoogleSearch

The definition of the doGoogleSearch method of the GoogleSearchPort interface is as follows:

```
function doGoogleSearch(const key:
    String; // your own Google Search Key
    const q: String; // query string
    const start: Integer; // start URLs
    const maxResults: Integer; // maximum
        // results
    const filter: Boolean; // filter alike
        // results?
    const restrict: String; // restrictions
    const safeSearch: Boolean; // adult
        // filter?
    const lr: String; // language?
    const ie: String; // input and output
        // encoding
    const oe: String): GoogleSearchResult;
stdcall;
```

Ouch! A lot of arguments, that's for sure. Fortunately, the Google Search API ZIP-file that we downloaded earlier also contains a file APIs_Reference.html (of 100,417 bytes) containing more information about the search request formats, and search results formats. Including the meaning of the arguments to doGoogleSearch.

The key argument is the Google Search Key that you have to obtain (in this article I will use the key 1WpiIaxr+k+hbyYbRLZOJfg7X9NgI837), the q argument is the actual query (there's a subsection on the complete query syntax, which includes the *site:* option to specify that you want to search within a specific website). The Start argument specifies where you want to start the results, and maxResults specifies how many results you want to receive (with a maximum of 10). Since you can only get a maximum of 10 results at a given time, Start can be used to specify where to start. If Start is 0, then you get the first 10 results.

To get the next 10 results, you must pass a value of 10 in Start, and so on. This will quickly consume your 1,000 available daily queries, so be aware not to use this to obtain all 10,000 results for "Dr.Bob" on the web. Personally, I think the first 10 results are just fine, so I use 0 for Start and 10 for maxResults. The Filter argument can be used to filter results that are "very similar", something that I also often use at Google myself, so I pass True as value for Filter. The Restrict argument can be used to restrict the search query to a specific country or topic within Google. The safeSearch argument can be set to True to make sure you don't get any "adult" search results. Handy if you want to build your own custom search engine for your kids at home (although I haven't tested this fully to make sure it really works as advertised). The lr argument is a bit similar to the restrict argument, and can be used to select results in a specific language (lr), such as Dutch or English (there seem to be no distinction between English, American English or any of the other English dialects). Finally, the ie and oe arguments specify the Input and Output Encoding, which can be set to latin1 for Dutch and English (see the reference document for more information).

In short, my call to doGoogleSearch, for a given Query string, would look as follows:

```
doGoogleSearch('1WpiIaxr+k+hbyYbRLZOJfg7X9NgI837',
    Query, 0, 10, True, '', True, 'lang_en',
    'latin1', 'latin1');
```

And this would give us a result of type `GoogleSearchResult`, which is derived from `TRemoteable` and the topic of our next section.

GoogleSearchResult

The `GoogleSearchResult` has a number of useful properties, such as `estimatedTotalResultsCount`, `searchTime`, and `resultElements`. The last one is an array, of which the elements are of type `ResultElement`, having a number of interesting subproperties such as `title`, `URL` and `cachedSize`. Using a `StringGrid` to display the results, the code to call `doGoogleSearch` and process the results is as follows:

```
procedure TForm1.btnSearchClick(Sender:
  TObject);
var
  Results: GoogleSearchResult;
  i: Integer;
begin
  Results :=
    GetGoogleSearchPort.doGoogleSearch(
      '1WpiIaxr+k+hbyYbRLZOJfg7X9NgI837',
      edtQuery.Text, 0, 10, True, '', True,
      'lang_en', 'latin1', 'latin1');
  Caption := Format('%d results in %.2n
    seconds',
    [Results.estimatedTotalResultsCount,
    Results.searchTime]);
  for i:=Low(Results.resultElements) to
    High(Results.resultElements) do
  begin
    StringGrid1.Cells[0,Succ(i)] :=
      IntToStr(Succ(i));
    StringGrid1.Cells[1,Succ(i)] :=
      Results.resultElements[i].title;
    StringGrid1.Cells[2,Succ(i)] :=
      Results.resultElements[i].URL;
    StringGrid1.Cells[3,Succ(i)] :=
      Results.resultElements[i].cachedSize
  end
end;
```

The result is a Windows application that can be used to enter a number of search words, and return the top 10 URLs. In order to jump directly to one of the resulting URLs, we only have to implement the `OnDblClick` event handler of the `StringGrid`, as follows:

```
procedure TForm1.StringGrid1DblClick(Sender:
  TObject);
begin
  with (Sender as TStringGrid) do
    ShellExecute(Handle, 'open', PChar(Cells[2,Row]), nil,
      nil, SW_NORMAL)
end;
```

The best thing is that you can integrate this feature in your own (non-commercial) applications as well, of course. As long as an internet connection is available to talk to Google's official Search web service.

#		URL	KB
1	Welcome to the UK Borland User Group	http://www.richkux.co.uk/	27k
2	UK-BUG - Member Purchasing	http://www.richkux.co.uk/html/party.asp	70k
3	UK-BUG - Borland User Group	http://www.drbob42.com/uk-bug/home.htm	14k
4	Dr Bob's UK-BUG - Borland User Group	http://www.drbob42.com/uk-bug/	2k
5	UK-BUG - Linux/Kylix Master Class	http://community.borland.com/article/11410.253/	70k
6	Bug Bug Register	http://www.3int.co.uk/bug/	34k
7	BPharmix News - UK-BUG - Developers	http://www.bphormix.com/btp_news_wf3.html	20k
8	BBC News UK-BUG - head denies	http://www.bbc.co.uk/1/english/uk/newsid_2340_2340	27k
9	pkugn.org.uk	http://pkugn.org.uk/	8k
10	dooyoo.com/uk/bug/ - weather's PC	http://www.dooyoo.co.uk/users/57044.html	47k

Apart from searching for keywords in a Windows GUI application (or added as a dialog to your own application), another good use of this functionality could be to add it to a website, transformed into a web server application. And in that case, you can prefix the Query text with the "site:" keyword, including the name of the website you're looking at. For example "site:www.drbob42.com" to look for the keyword in pages on my own website. As an example of the output, take a look at the figure below which shows a search for SOAP in the site:www.drbob42.com.

#		URL	KB
1	Dr Bob's SOAP - Bubbles	http://www.drbob42.com/soap/	7k
2	Dr Bob's Soap - SOAP - Soap, DataSnap	http://www.drbob42.com/soap42.htm	9k
3	Dr Bob's SOAP - Bubbles with XML - 5Dr	http://www.drbob42.com/soap/home.htm	54k
4	Dr Bob's Dynamic Soap - SOAP - Soap, D	http://www.drbob42.com/soap/soap42.htm	7k
5	Dr Bob's Kyle 2 Developer's Guide Soap chapter	http://www.drbob42.com/kyle/soap0.htm	12k
6	Detailed Book Review: Professional XML Web Ser	http://www.drbob42.com/reviews/186100593.htm	17k
7	Dr Bob's Kyle 2 Developer's Guide Soap chapter	http://www.drbob42.com/kyle/soap0.htm	15k
8	Dr Bob Examines...	http://www.drbob42.com/dq/dq5/examines.htm	15k
9	Dr Bob's Kyle 2 Developer's Guide Soap chapter	http://www.drbob42.com/kyle/soap0.htm	13k
10	Dr Bob's Kyle 2 Developer's Guide Soap chapter	http://www.drbob42.com/kyle/soap1.htm	24k

Conclusion

In this article, I've tried to explain how to consume Web Services with Delphi 6. We've seen how to generate the import files based on a WSDL definition of the Web Service, and how to actually call the methods from the Web Service. For more information about SOAP and Web Services in C++Builder, Delphi, Kylix or JBuilder, check out my SOAP Bubbles website at <http://www.drbob42.com/soap/>, where you can also download full source code for this Google42 web service client (including any enhancements that I may have added since the time of writing).



Bob Swart (aka Dr.Bob - www.drbob42.com) is an author, trainer and consultant who recently started his own one-man company called eBob42 in Helmond, The Netherlands. Bob, who writes his own Delphi Clinic training material, has spoken at Delphi and Borland Developer Conferences since 1993. Bob is co-author of the Revolutionary Guide to Delphi 2, Delphi 4 Unleashed, C++Builder 4 Unleashed, C++Builder 5 Developer's Guide, Kylix Developer's Guide, Delphi 6 Developer's Guide and the upcoming C++Builder 6 Developer's Guide.