Cameleon#[1] User Manual

Aykut FIRAT

24 May 2008

Working Paper CISL# 2008-13

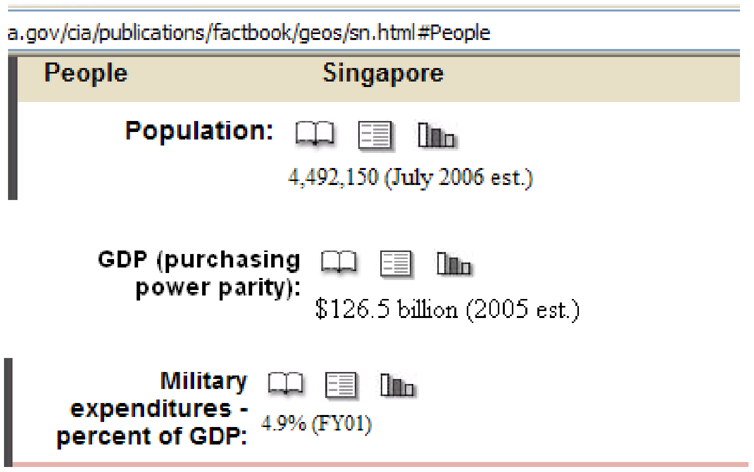
Contents

Composite Information Systems Laboratory (CISL)
Sloan School of Management, Room E53-320
Massachusetts Institute of Technology
Cambridge, MA 02142

[1] **Cameleon#** refers to **Cameleon# Server** and **Cameleon# Studio**

What is Cameleon# Server?

Cameleon# Server is a server application coded in C# to extract data from web sites based on a specification file. For example, the following information that can be found in the CIA World Fact-book can be extracted in table or XML format by first 1) authoring a specification (spec) file for that web site, and then 2) sending a query to Cameleon.


a.gov/cia/publications/factbook/geos/sn.html#People

People Singapore

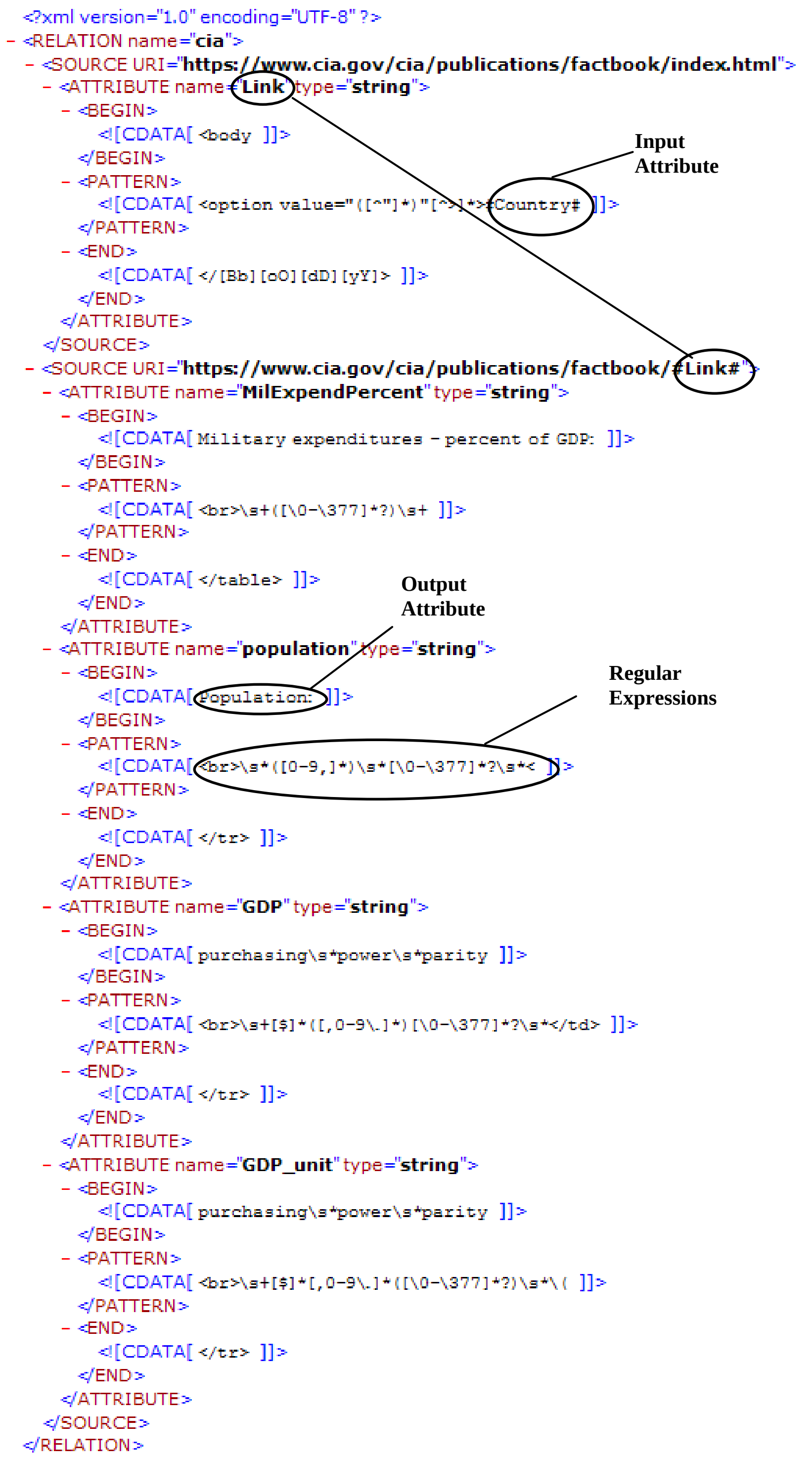
Population: 4,492,150 (July 2006 est.)

GDP (purchasing power parity): $126.5 billion (2005 est.)

Military expenditures - percent of GDP: 4.9% (FY01)

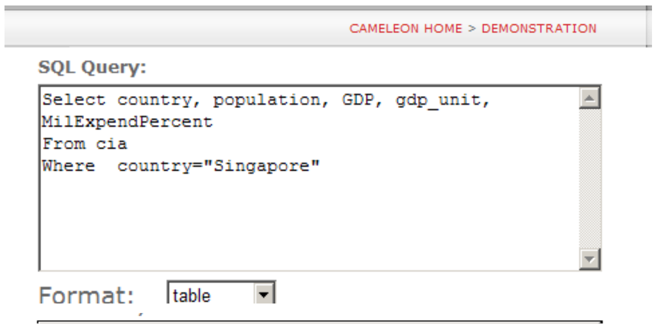
Figure 1. Available data in CIA World Fact Book site

The specification (spec) file for extracting data from the CIA factbook web site is shown below. This spec file can be authored visually or manually. After the spec file is completed a query in simplified SQL form can be sent to the Cameleon server via the http protocol and results are returned as shown in Figure 3.

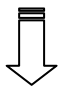


```
  <?xml version="1.0" encoding="UTF-8" ?>
- <RELATION name="cia">
  - <SOURCE URI="https://www.cia.gov/cia/publications/factbook/index.html">
    - <ATTRIBUTE name="Link" type="string">
      - <BEGIN>
          <![CDATA[ <body ]]>
        </BEGIN>
      - <PATTERN>
          <![CDATA[ <option value="([^"]*)"[^>]*>#Country# ]]>
        </PATTERN>
      - <END>
          <![CDATA[ </[Bb][oO][dD][yY]> ]]>
        </END>
      </ATTRIBUTE>
    </SOURCE>
  - <SOURCE URI="https://www.cia.gov/cia/publications/factbook/#Link#">
    - <ATTRIBUTE name="MilExpendPercent" type="string">
      - <BEGIN>
          <![CDATA[ Military expenditures - percent of GDP: ]]>
        </BEGIN>
      - <PATTERN>
          <![CDATA[ <br>\s+([\0-\377]*?)\s+ ]]>
        </PATTERN>
      - <END>
          <![CDATA[ </table> ]]>
        </END>
      </ATTRIBUTE>
    - <ATTRIBUTE name="population" type="string">
      - <BEGIN>
          <![CDATA[ Population: ]]>
        </BEGIN>
      - <PATTERN>
          <![CDATA[ <br>\s*([0-9,]*)\s*[\0-\377]*?\s*< ]]>
        </PATTERN>
      - <END>
          <![CDATA[ </tr> ]]>
        </END>
      </ATTRIBUTE>
    - <ATTRIBUTE name="GDP" type="string">
      - <BEGIN>
          <![CDATA[ purchasing\s*power\s*parity ]]>
        </BEGIN>
      - <PATTERN>
          <![CDATA[ <br>\s+[$]*([,0-9\.]*)[\0-\377]*?\s*</td> ]]>
        </PATTERN>
      - <END>
          <![CDATA[ </tr> ]]>
        </END>
      </ATTRIBUTE>
    - <ATTRIBUTE name="GDP_unit" type="string">
      - <BEGIN>
          <![CDATA[ purchasing\s*power\s*parity ]]>
        </BEGIN>
      - <PATTERN>
          <![CDATA[ <br>\s+[$]*[,0-9\.]*([\0-\377]*?)\s*\( ]]>
        </PATTERN>
      - <END>
          <![CDATA[ </tr> ]]>
        </END>
      </ATTRIBUTE>
    </SOURCE>
  </RELATION>
```

Figure 2. Example Specification File for the CIA Web Source



```
Select country, population, GDP, gdp_unit,
MilExpendPercent
From cia
Where  country="Singapore"
```



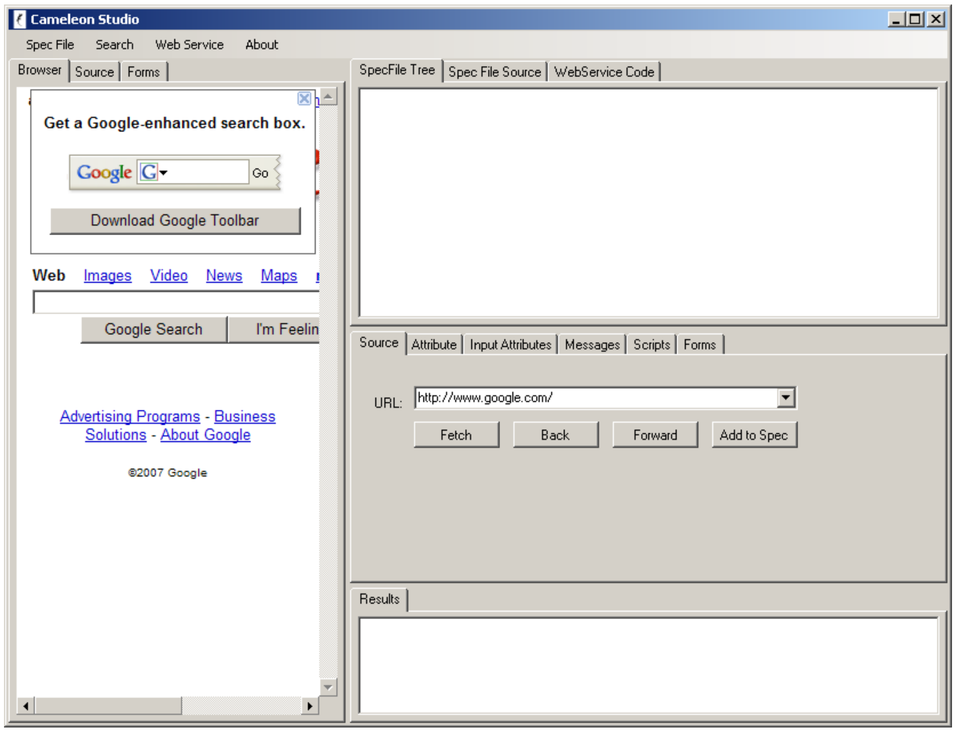
country	population	gdp	gdp_unit	milexpendpercent
Singapore	4492150	126.5	billion	4.90

Figure 3. Simple SQL Query against the wrapped CIA World Fact Book

What is Cameleon# Studio?

Cameleon# Studio is a visual application that aids the development of spec files, and converting them into web services.

It has a built in browser on the left that also shows the source of a web page, and the forms that are in that web page. On the upper right it shows the spec file in tree form, and original form, and the auto produced web service code. On the lower right, it has several tabs for surfing web sites (Sources), defining attributes (Attributes), providing values for input attributes (Input Attributes), displaying messages from the program (Messages), displaying scripts from the web sites, and authoring custom forms (forms). Results can also be viewed via the Results tab.



URL: http://www.google.com/

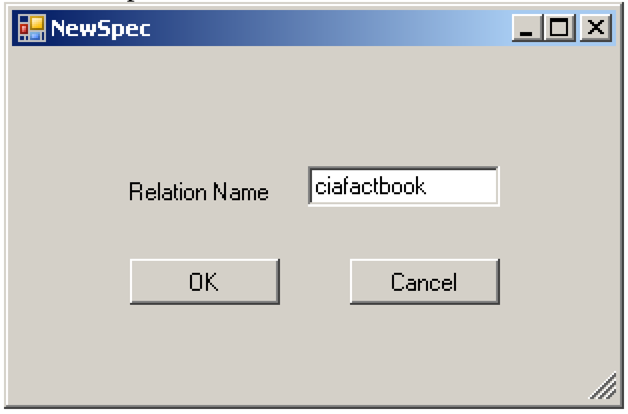
Figure 4. Cameleon Studio Interface

Cameleon# Studio by Example

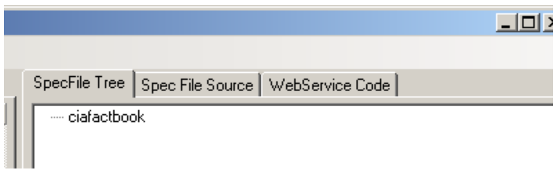
In this section we go through an example by authoring a specification file for the CIA World Fact Book using Cameleon# Studio.

Step 1 – Give a name

Use the Spec File menu item, click on New and enter a spec name. (e.g. ciafactbook)

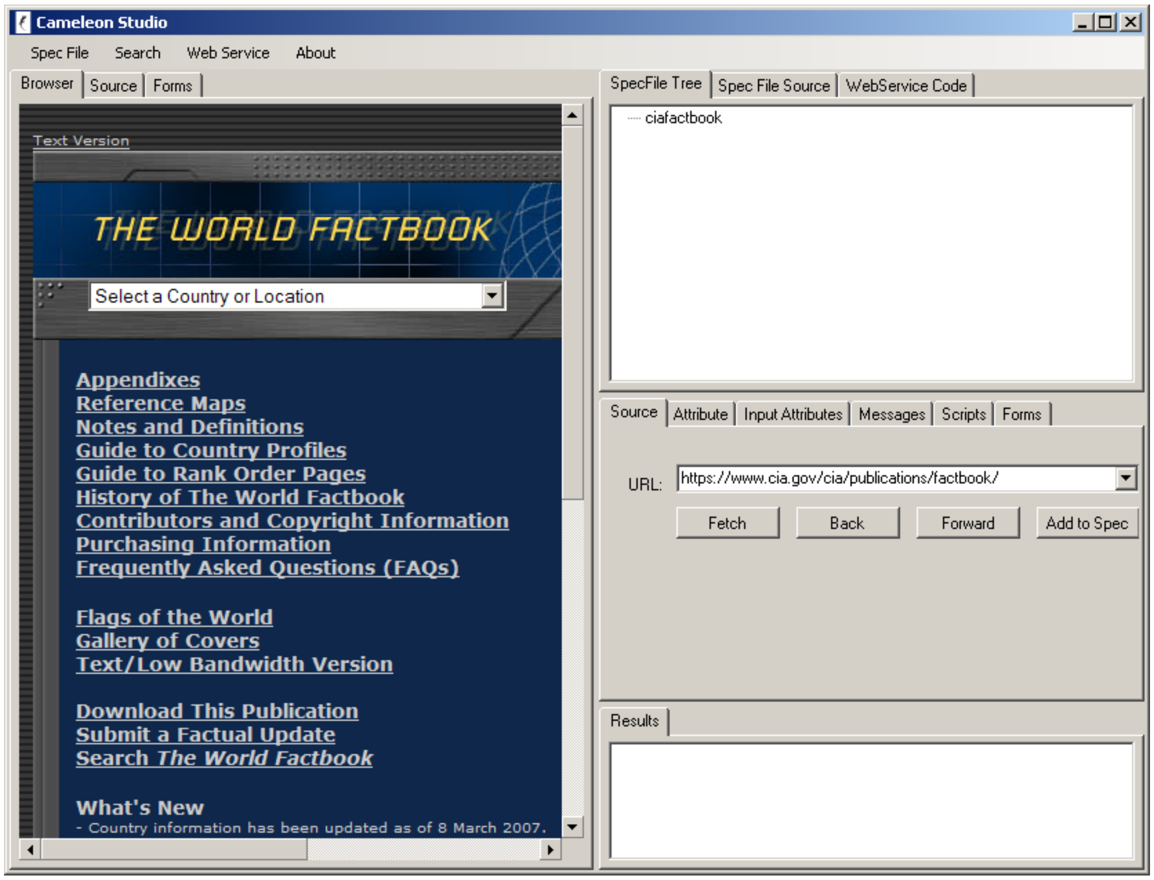


Relation Name: ciafactbook

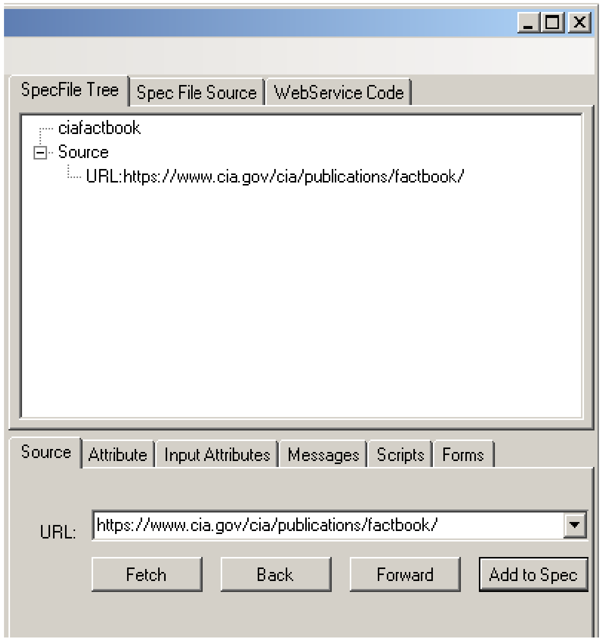


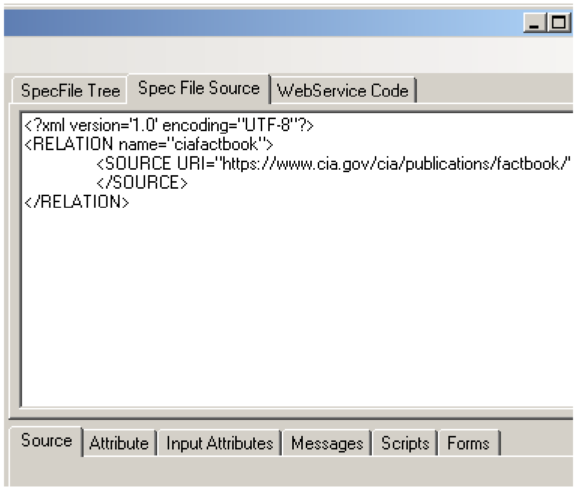
Step 2 – Go to the web site

Either type it directly in the URL text box in the Source tab on the right, or search it in Google and find it.



We are trying to simulate how the extraction should happen, and this is the page where we start. We click on Add to Spec button in the Source tab.





```
<?xml version='1.0' encoding="UTF-8"?>
<RELATION name="ciafactbook">
        <SOURCE URI="https://www.cia.gov/cia/publications/factbook/"
        </SOURCE>
</RELATION>
```

If you click on the Spec File Source tab you will see that spec file in XML format is being constructed automatically.

Step 3 Define an Attribute

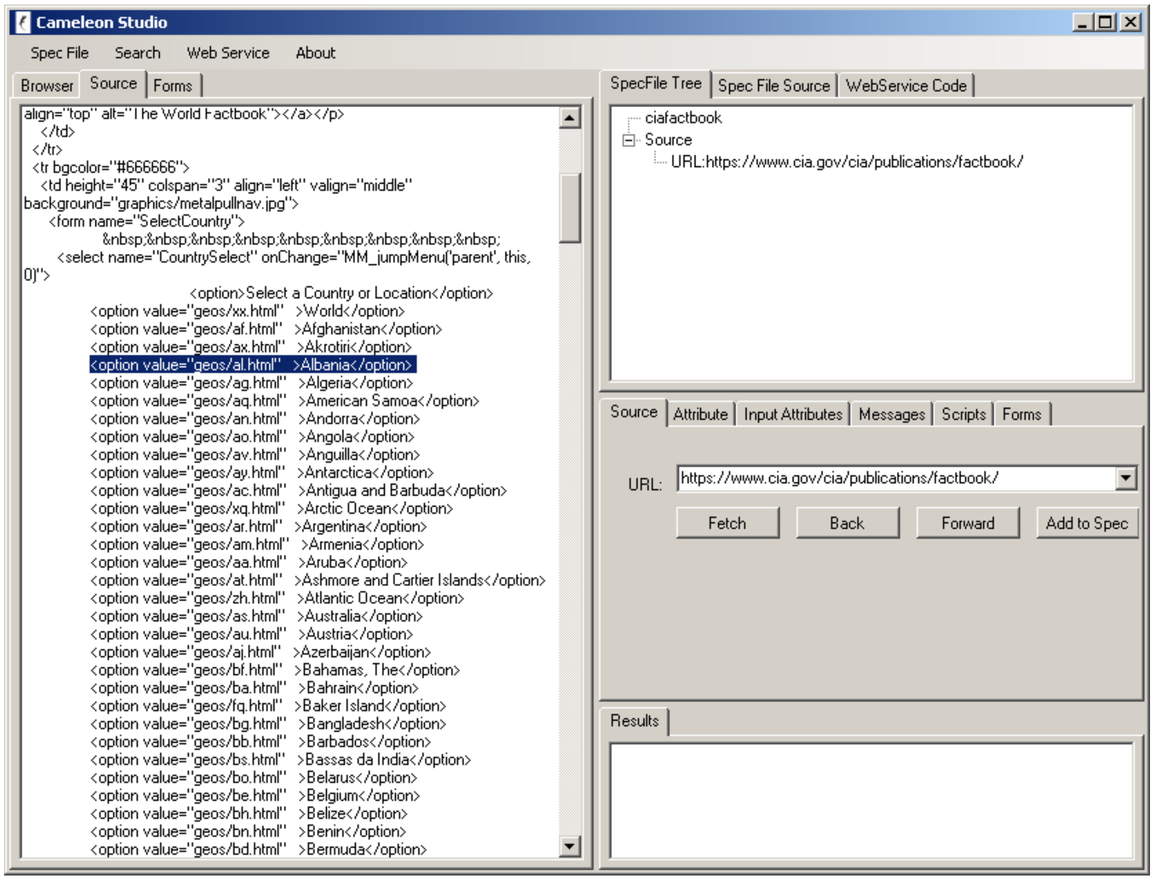
Our first attribute will be the link that will take us to the page that has the information. If you view the source of the document you will notice that relative links for each country have the following structure:

```
<option value="geos/al.html"   >Albania</option>
```

Or if we generalize:

```
<option value="#Link#"   >#Country#</option>
```

Where #Country# is an input country and the #Link# is the corresponding link.

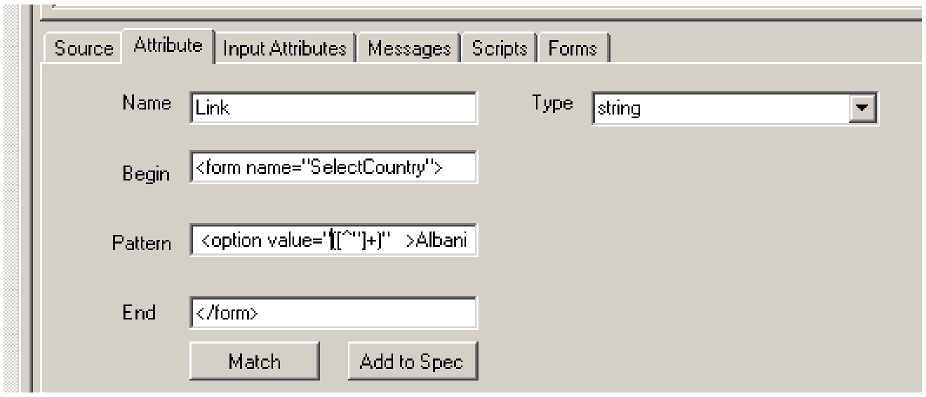


```
align="top" alt="The World Factbook"></a></p>
   </td>
  </tr>
  <tr bgcolor="#666666">
   <td height="45" colspan="3" align="left" valign="middle"
background="graphics/metalpullnav.jpg">
    <form name="SelectCountry">
          &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;
     <select name="CountrySelect" onChange="MM_jumpMenu('parent', this,
0)">
                    <option>Select a Country or Location</option>
          <option value="geos/xx.html"   >World</option>
          <option value="geos/af.html"   >Afghanistan</option>
          <option value="geos/ax.html"   >Akrotiri</option>
          <option value="geos/al.html"   >Albania</option>
          <option value="geos/ag.html"   >Algeria</option>
          <option value="geos/aq.html"   >American Samoa</option>
          <option value="geos/an.html"   >Andorra</option>
          <option value="geos/ao.html"   >Angola</option>
          <option value="geos/av.html"   >Anguilla</option>
          <option value="geos/ay.html"   >Antarctica</option>
          <option value="geos/ac.html"   >Antigua and Barbuda</option>
          <option value="geos/xq.html"   >Arctic Ocean</option>
          <option value="geos/ar.html"   >Argentina</option>
          <option value="geos/am.html"   >Armenia</option>
          <option value="geos/aa.html"   >Aruba</option>
          <option value="geos/at.html"   >Ashmore and Cartier Islands</option>
          <option value="geos/zh.html"   >Atlantic Ocean</option>
          <option value="geos/as.html"   >Australia</option>
          <option value="geos/au.html"   >Austria</option>
          <option value="geos/aj.html"   >Azerbaijan</option>
          <option value="geos/bf.html"   >Bahamas, The</option>
          <option value="geos/ba.html"   >Bahrain</option>
          <option value="geos/fq.html"   >Baker Island</option>
          <option value="geos/bg.html"   >Bangladesh</option>
          <option value="geos/bb.html"   >Barbados</option>
          <option value="geos/bs.html"   >Bassas da India</option>
          <option value="geos/bo.html"   >Belarus</option>
          <option value="geos/be.html"   >Belgium</option>
          <option value="geos/bh.html"   >Belize</option>
          <option value="geos/bn.html"   >Benin</option>
          <option value="geos/bd.html"   >Bermuda</option>
```

ciafactbook
Source
URL:https://www.cia.gov/cia/publications/factbook/

URL: https://www.cia.gov/cia/publications/factbook/

Click on the attribute tab and enter the info in the boxes as follows:



Name: Link
Type: string
Begin: <form name="SelectCountry">
Pattern: <option value="([^"]+)"   >Albani
End: </form>

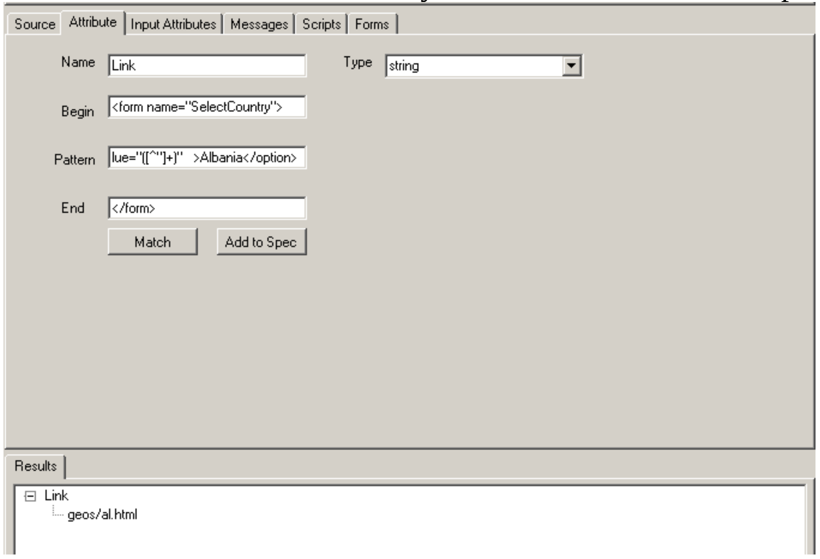
Here Link is the name of our attribute

<form name="SelectCountry"> and </form> designates the begin and end of the region we are interested in this page via regular expressions. Pattern is the regular expression denoting what to extract from this region. In this case the pattern[2] is <option value="([^"]+)"   >#Country#</option>

But we will use an actual country name instead of the input parameter #Country# until we are satisfied with our attribute definition for testing purposes.
(i.e. it will be: <option value="([^"]+)"   >Albania</option>).
The attribute text boxes have a couple of hidden functionalities:

1) If you double click on the Begin box, it will eliminate anything before the first match of that pattern in the Source box.
2) If you double click on the End box, it will eliminate anything after the first match of that pattern.
3) If you double click on the Pattern box, it will highlight the text it matches in the text in blue. If there is no match, then there will no highlighting. Note that if the pattern matches more than once then multiple highlights will be shown.
4) If you double click on the Name box, it will restore the Source box into its original form.
5) Note also that you can drag (not drag and drop unfortunately so be careful as it will take the value as soon as you enter a text box, this is due to some bug in .NET) some text from the text into attribute boxes.

Click on match and it will show you the match in the results pane:

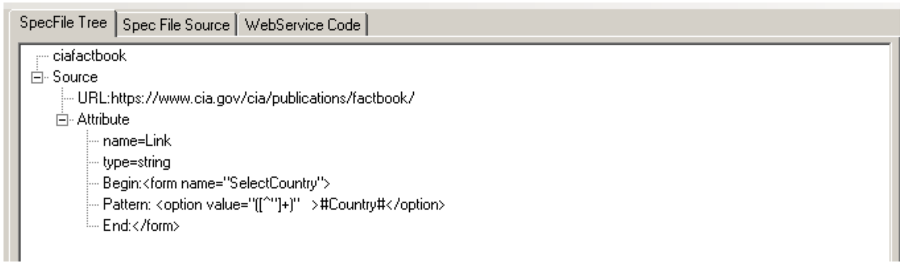


Name: Link
Type: string
Begin: <form name="SelectCountry">
Pattern: lue="([^"]+)"   >Albania</option>
End: </form>

Link
geos/al.html

This is what we want. Change now Albania into #Country# so that it will work for all Countries and click on Add to Spec.

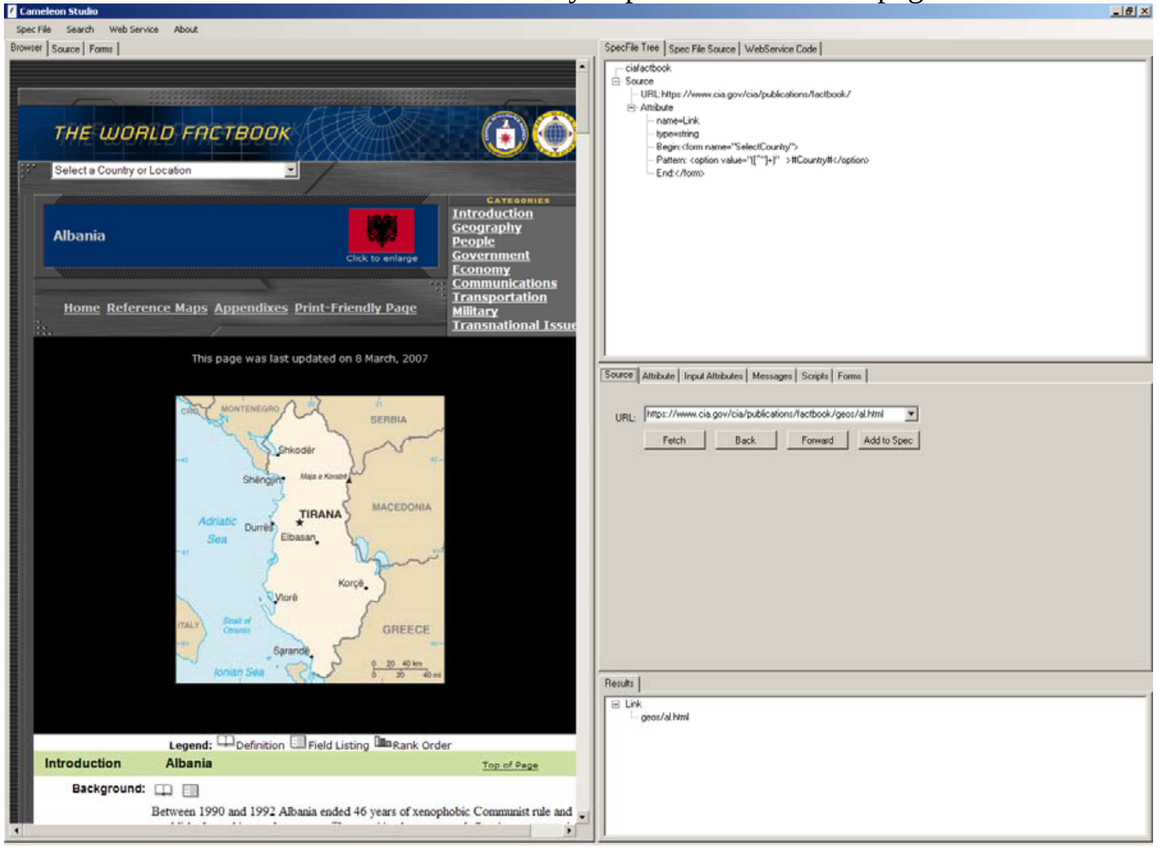
[2] Please see the Cameleon# user manual at the end of this manual if you have difficulty understanding what this pattern means.



```
ciafactbook
Source
    URL:https://www.cia.gov/cia/publications/factbook/
    Attribute
        name=Link
        type=string
        Begin:<form name="SelectCountry">
        Pattern: <option value="([^"]+)"  >#Country#</option>
        End:</form>
```

Step 4 Define the Second Source

Go back to the web site and choose a country to proceed to the next page.



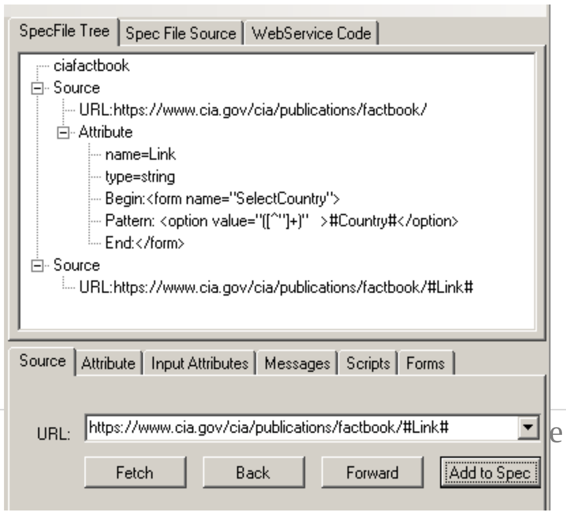
Between 1990 and 1992 Albania ended 46 years of xenophobic Communist rule and

```
ciafactbook
Source
    URL:https://www.cia.gov/cia/publications/factbook/
    Attribute
        name=Link
        type=string
        Begin:<form name="SelectCountry">
        Pattern: <option value="([^"]+)"  >#Country#</option>
        End:</form>
```

URL: https://www.cia.gov/cia/publications/factbook/geos/al.html

```
Link
    geos/al.html
```

Note that the address in the Source tab has a fixed part and a part that varies based on a selected country. This variable part is stored in our attribute Link. To go to this site we can simply append the Link attribute to the fixed part of the address. Change the variable part of the address in the Source text box and add it to spec.

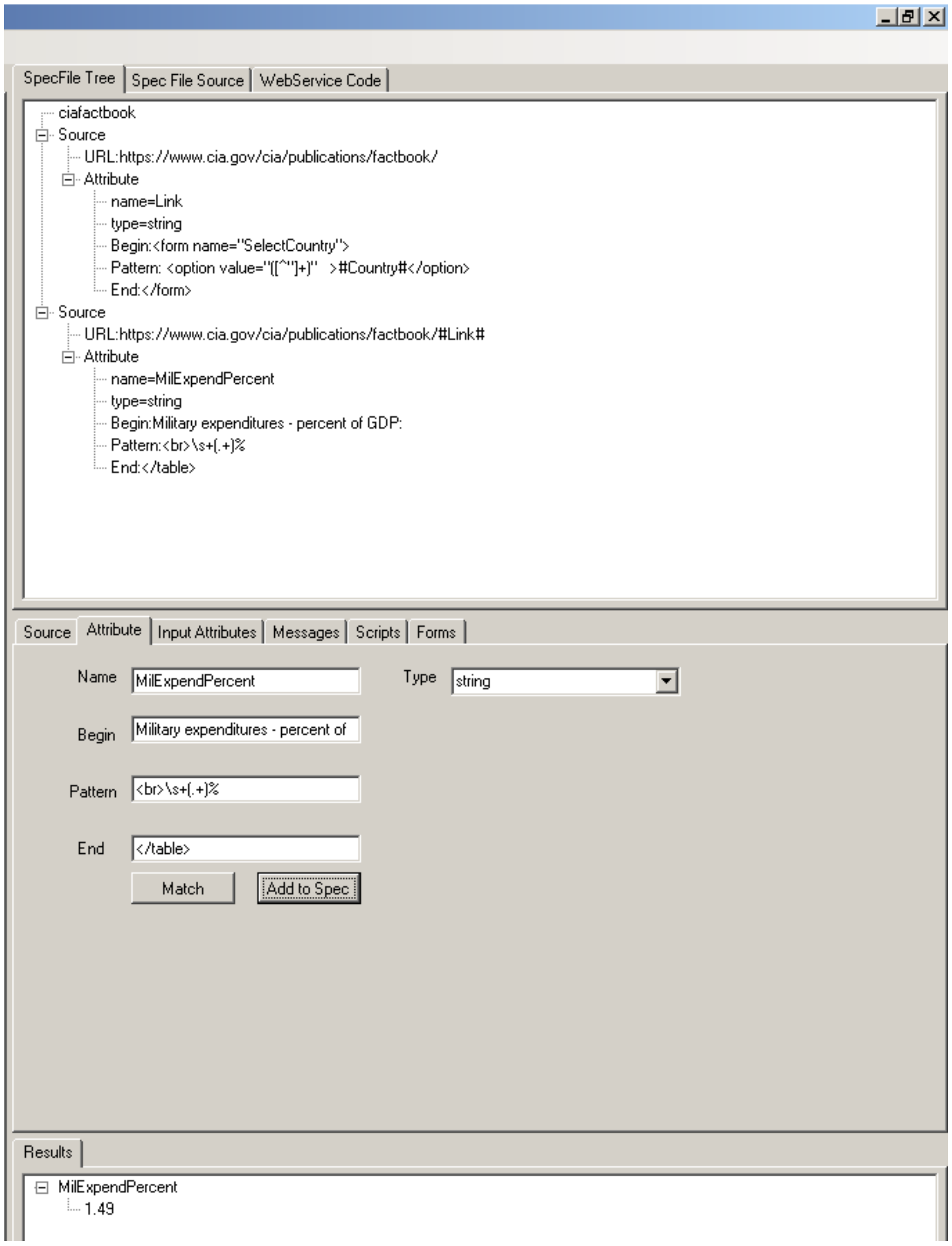


```
ciafactbook
Source
    URL:https://www.cia.gov/cia/publications/factbook/
    Attribute
        name=Link
        type=string
        Begin:<form name="SelectCountry">
        Pattern: <option value="([^"]+)"  >#Country#</option>
        End:</form>
Source
    URL:https://www.cia.gov/cia/publications/factbook/#Link#
```

URL: https://www.cia.gov/cia/publications/factbook/#Link#

Step 5 Define more attributes

Note that before you add an attribute make sure you click on the source in the spec tree that that attribute belongs to. Otherwise the attribute may be added to the wrong source. In that case right click on the attribute and choose delete to remove the attribute.



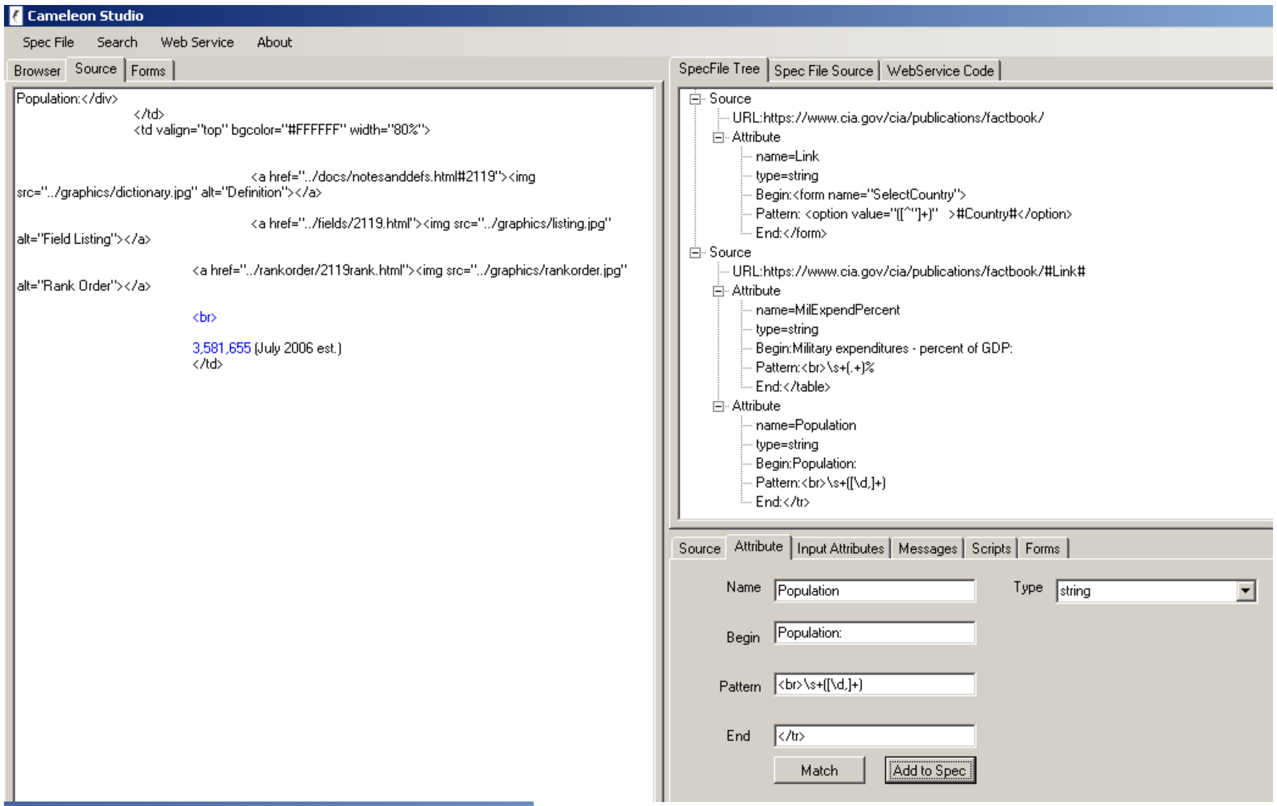
```
ciafactbook
Source
    URL:https://www.cia.gov/cia/publications/factbook/
    Attribute
        name=Link
        type=string
        Begin:<form name="SelectCountry">
        Pattern: <option value="([^"]+)"  >#Country#</option>
        End:</form>
Source
    URL:https://www.cia.gov/cia/publications/factbook/#Link#
    Attribute
        name=MilExpendPercent
        type=string
        Begin:Military expenditures - percent of GDP:
        Pattern:<br>\s+(.+)%
        End:</table>
```

Name: MilExpendPercent
Type: string
Begin: Military expenditures - percent of
Pattern:
\s+(.+)%
End: </table>

```
MilExpendPercent
    1.49
```

When you author attributes you can use the search menu item to find what you are looking for in the web site or in its source.

Population, GDP and GDPUnit attributes are declared as follows:



Cameleon Studio

Spec File | Search | Web Service | About

Browser | Source | Forms

```
Population:</div>
                    </td>
                    <td valign="top" bgcolor="#FFFFFF" width="80%">


                                        <a href="../docs/notesanddefs.html#2119"><img
src="../graphics/dictionary.jpg" alt="Definition"></a>

                                        <a href="../fields/2119.html"><img src="../graphics/listing.jpg"
alt="Field Listing"></a>

                              <a href="../rankorder/2119rank.html"><img src="../graphics/rankorder.jpg"
alt="Rank Order"></a>

                              <br>

                              3,581,655 (July 2006 est.)
                              </td>
```

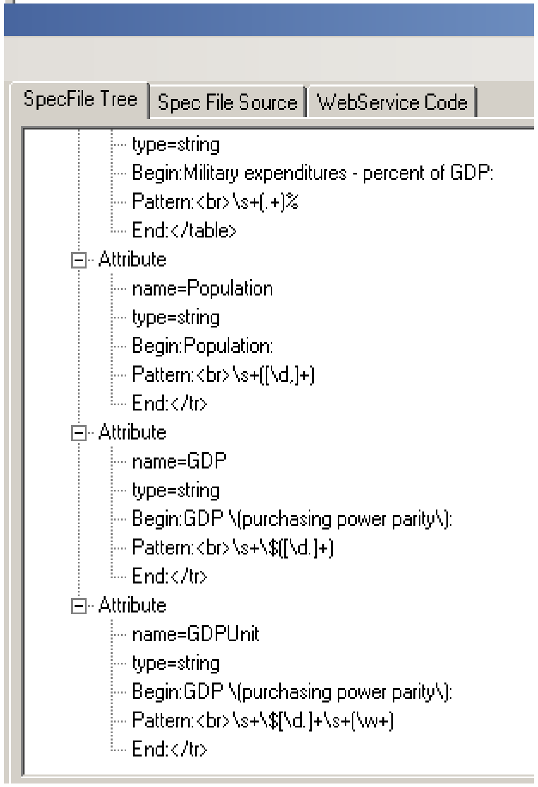
SpecFile Tree | Spec File Source | WebService Code

```
⊟ Source
   URL:https://www.cia.gov/cia/publications/factbook/
   ⊟ Attribute
      name=Link
      type=string
      Begin:<form name="SelectCountry">
      Pattern: <option value="([^"]+)"   >#Country#</option>
      End:</form>
⊟ Source
   URL:https://www.cia.gov/cia/publications/factbook/#Link#
   ⊟ Attribute
      name=MilExpendPercent
      type=string
      Begin:Military expenditures - percent of GDP:
      Pattern:<br>\s+(.+)%
      End:</table>
   ⊟ Attribute
      name=Population
      type=string
      Begin:Population:
      Pattern:<br>\s+([\d,]+)
      End:</tr>
```

Source | Attribute | Input Attributes | Messages | Scripts | Forms

Name	Population	Type	string
Begin	Population:		
Pattern	 \s+([\d,]+)		
End	</tr>		

Match | Add to Spec

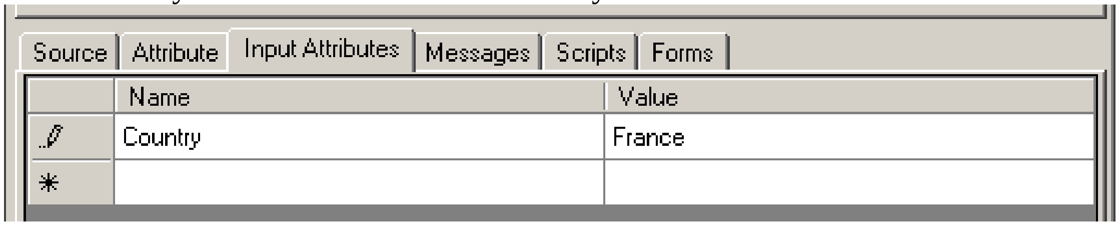


SpecFile Tree | Spec File Source | WebService Code

```
      type=string
      Begin:Military expenditures - percent of GDP:
      Pattern:<br>\s+(.+)%
      End:</table>
   ⊟ Attribute
      name=Population
      type=string
      Begin:Population:
      Pattern:<br>\s+([\d,]+)
      End:</tr>
   ⊟ Attribute
      name=GDP
      type=string
      Begin:GDP \(purchasing power parity\):
      Pattern:<br>\s+\$([\d.]+)
      End:</tr>
   ⊟ Attribute
      name=GDPUnit
      type=string
      Begin:GDP \(purchasing power parity\):
      Pattern:<br>\s+\$[\d.]+\s+(\w+)
      End:</tr>
```

Step 6 Define the Input Attributes

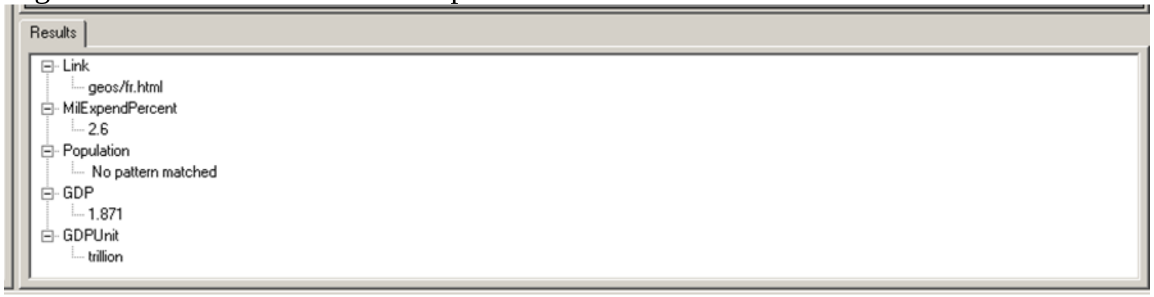
Click on Input Attributes box and enter Country and a value for a country. Note that these attributes can either be obtained from the query or may be extracted from a source. If they are extracted from a source they can be used in consecutive sources.



	Name	Value
	Country	France

Step 7. Test

Right Click on ciafactbook in the specfile tree and choose Run.

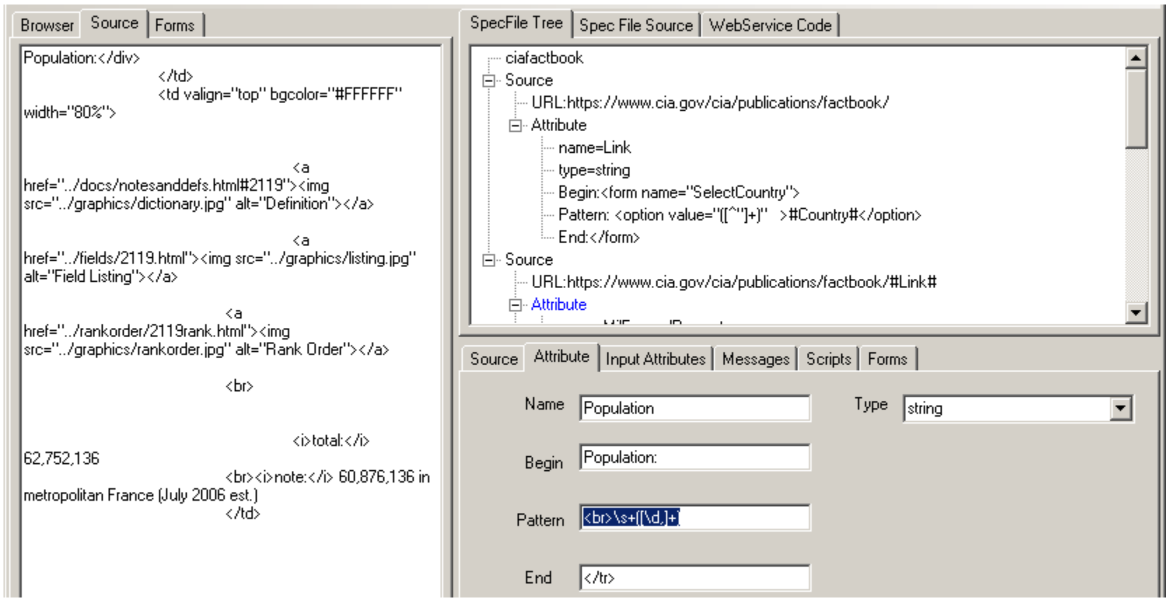


```
Link
  geos/fr.html
MilExpendPercent
  2.6
Population
  No pattern matched
GDP
  1.871
GDPUnit
  trillion
```

It seems everything is fine except Population has a problem.

Step 8. Debug

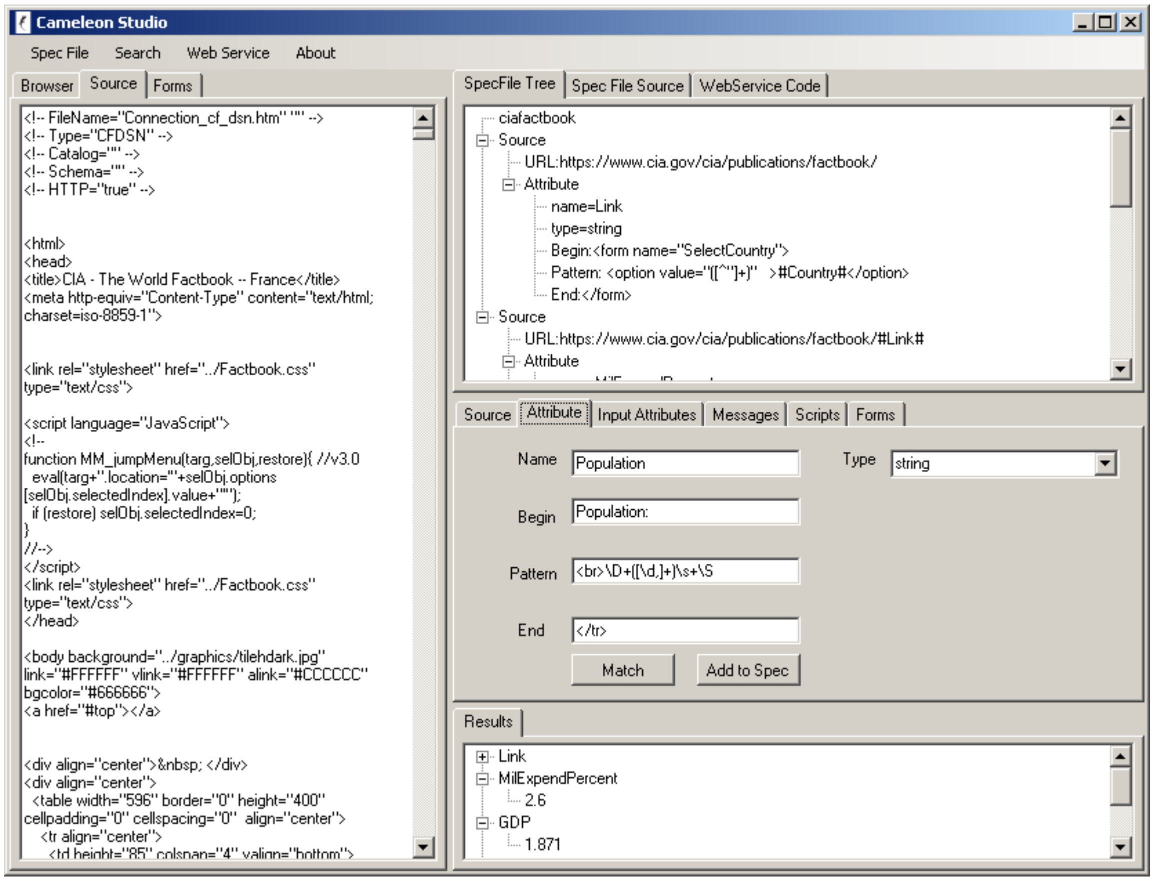
Right Click on ciafactbook in the specfile tree and choose Step. Click on step a couple of times until you arrive to the second source. Then switch to attribute tab, right click on the attribute Population from the specfile tree and click on Transfer Attribute – this will fill in the fields in the lower half of the screen. You will notice if you double click on the begin first, and end later, and then pattern that the pattern indeed fails, because population is not consistently expressed with the pattern that we found in Albania. We need to look for a more common pattern. (Note that you can also put a breakpoint somewhere in your spec file so that it runs until the breakpoint by right clicking on the item of interest in the spec file tree. You can also remove the break points by using delete break point.)


```
Population:</div>
                    </td>
                    <td valign="top" bgcolor="#FFFFFF"
width="80%">


                                        <a
href="../docs/notesanddefs.html#2119"><img
src="../graphics/dictionary.jpg" alt="Definition"></a>

                                        <a
href="../fields/2119.html"><img src="../graphics/listing.jpg"
alt="Field Listing"></a>

                              <a
href="../rankorder/2119rank.html"><img
src="../graphics/rankorder.jpg" alt="Rank Order"></a>

                              <br>


                                        <i>total:</i>
62,752,136
                              <br><i>note:</i> 60,876,136 in
metropolitan France (July 2006 est.)
                              </td>
```

```
ciafactbook
Source
    URL:https://www.cia.gov/cia/publications/factbook/
    Attribute
        name=Link
        type=string
        Begin:<form name="SelectCountry">
        Pattern: <option value="([^"]+)"   >#Country#</option>
        End:</form>
Source
    URL:https://www.cia.gov/cia/publications/factbook/#Link#
    Attribute
```

Name	Population
Type	string
Begin	Population:
Pattern	 \s+([\d,]+)
End	</tr>

The following one works for both France and Albania so presumably will work for others as well. Delete the existing attribute (right click, choose delete) and put the attribute declaration. Test again with other countries.


```
<!-- FileName="Connection_cf_dsn.htm" "" -->
<!-- Type="CFDSN" -->
<!-- Catalog="" -->
<!-- Schema="" -->
<!-- HTTP="true" -->


<html>
<head>
<title>CIA - The World Factbook -- France</title>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1">


<link rel="stylesheet" href="../Factbook.css"
type="text/css">

<script language="JavaScript">
<!--
function MM_jumpMenu(targ,selObj,restore){ //v3.0
  eval(targ+".location='"+selObj.options
[selObj.selectedIndex].value+"'");
  if (restore) selObj.selectedIndex=0;
}
//-->
</script>
<link rel="stylesheet" href="../Factbook.css"
type="text/css">
</head>

<body background="../graphics/tilehdark.jpg"
link="#FFFFFF" vlink="#FFFFFF" alink="#CCCCCC"
bgcolor="#666666">
<a href="#top"></a>


<div align="center">&nbsp; </div>
<div align="center">
  <table width="596" border="0" height="400"
cellpadding="0" cellspacing="0"  align="center">
    <tr align="center">
```

```
ciafactbook
Source
    URL:https://www.cia.gov/cia/publications/factbook/
    Attribute
        name=Link
        type=string
        Begin:<form name="SelectCountry">
        Pattern: <option value="([^"]+)"   >#Country#</option>
        End:</form>
Source
    URL:https://www.cia.gov/cia/publications/factbook/#Link#
    Attribute
```

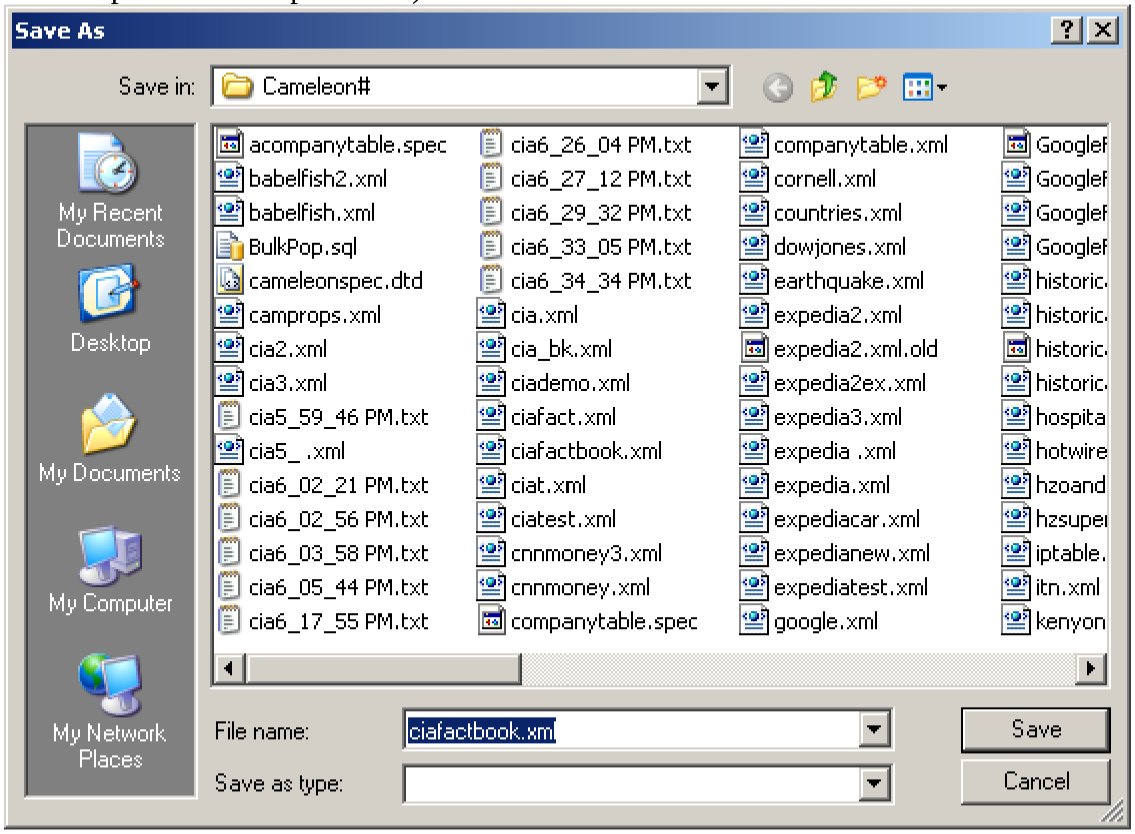
Name	Population
Type	string
Begin	Population:
Pattern	 \D+([\d,]+)\s+\S
End	</tr>

```
Link
MilExpendPercent
    2.6
GDP
    1.871
```

Step 9 Transfer

Now that we are satisfied, we can transfer our spec file to a location we can reach publicly. This can either be manually done, i.e. by saving it in your local directory and transferring via your favorite ftp application or using Cameleon# Studio.

Click on Spec File menu item and choose save by giving the same name specified when you first created the spec file with xml extension: (Giving a different name will create a problem for ftp transfer)

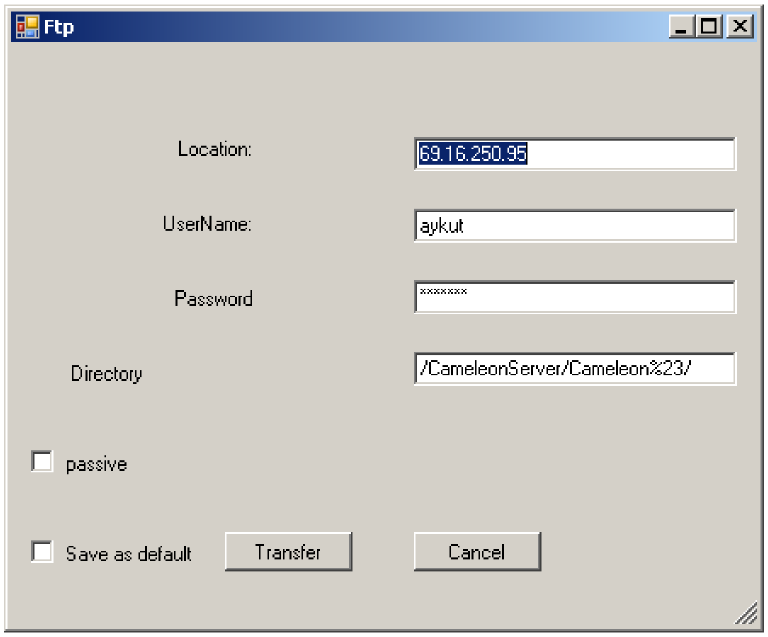


acompanytable.spec	cia6_26_04 PM.txt	companytable.xml	GoogleF
babelfish2.xml	cia6_27_12 PM.txt	cornell.xml	GoogleF
babelfish.xml	cia6_29_32 PM.txt	countries.xml	GoogleF
BulkPop.sql	cia6_33_05 PM.txt	dowjones.xml	GoogleF
cameleonspec.dtd	cia6_34_34 PM.txt	earthquake.xml	historic.
camprops.xml	cia.xml	expedia2.xml	historic.
cia2.xml	cia_bk.xml	expedia2.xml.old	historic.
cia3.xml	ciademo.xml	expedia2ex.xml	historic.
cia5_59_46 PM.txt	ciafact.xml	expedia3.xml	hospita
cia5_ .xml	ciafactbook.xml	expedia .xml	hotwire
cia6_02_21 PM.txt	ciat.xml	expedia.xml	hzoand
cia6_02_56 PM.txt	ciatest.xml	expediacar.xml	hzsupe
cia6_03_58 PM.txt	cnnmoney3.xml	expedianew.xml	iptable.
cia6_05_44 PM.txt	cnnmoney.xml	expediatest.xml	itn.xml
cia6_17_55 PM.txt	companytable.spec	google.xml	kenyon

File name: ciafactbook.xml

Save as type:

After saving click on Spec File menu item and Transfer. Below I show a location that I have privileges to upload my files. You should replace the boxes with your own information. Note that you can use passive ftp if you want or your server requires by clicking on the passive check box, and you can also save your login information by checking the Save as default box. Once the information is entered clicking on Transfer will ftp your spec file to a remote location. As mentioned before this step can be achieved manually as well.



Ftp

Location: 69.16.250.95

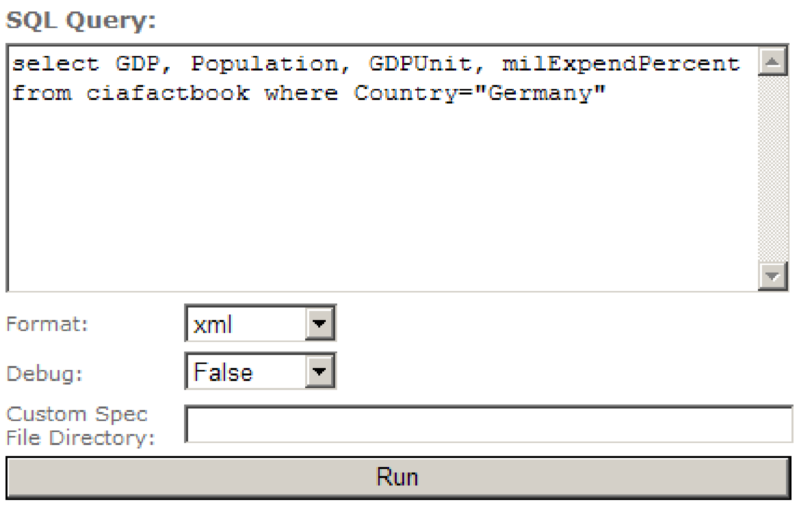
UserName: aykut

Password *******

Directory /CameleonServer/Cameleon%23/

Step 10 Test on the Web

Now go to a public Cameleon# test location. For example:
1) http://interchange.mit.edu/cameleon_sharp/cameleon.html
2.) http://www.aykutfirat.com/Cameleon/Demo.aspx
In these demo pages you can use a custom spec file location by entering a registry directory. For example if you put the spec file in your MIT www directory, use http://www.mit.edu/~username/ as your registry directory. (Of course replace username with your username)
Then enter the query shown below:



SQL Query:

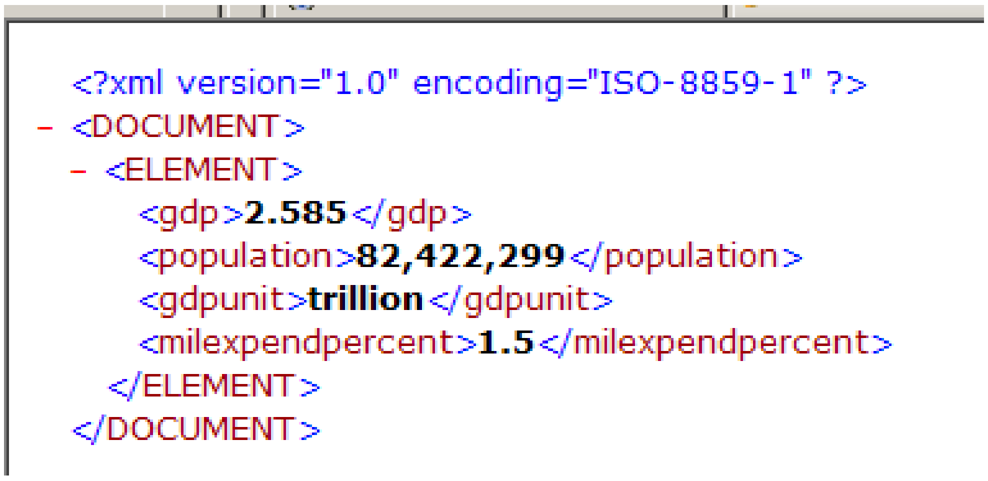
```
select GDP, Population, GDPUnit, milExpendPercent
from ciafactbook where Country="Germany"
```

Format: xml

Debug: False

Custom Spec File Directory:

Note that I directly put the spec file in the local directory so I leave the custom spec file directory empty. You should fill it appropriately pointing to the location of your spec file.
When you click on Run you should get:

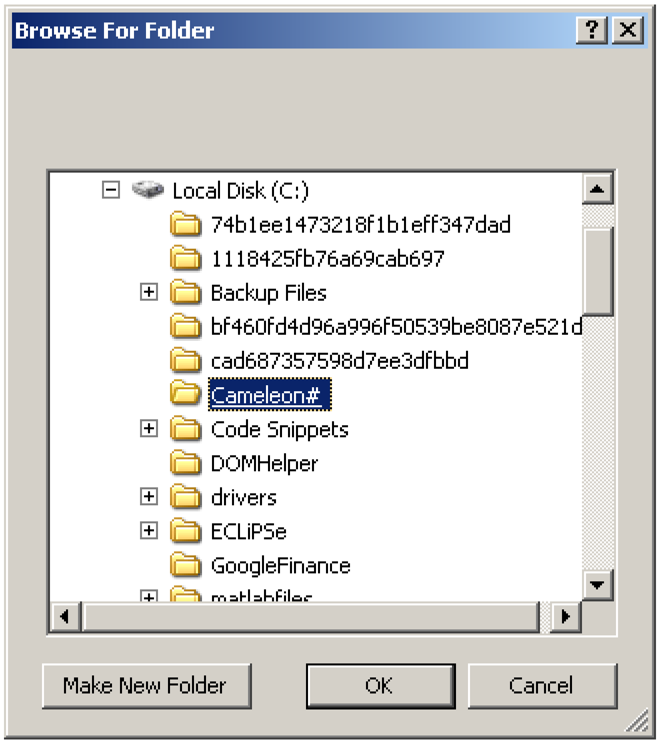


```
<?xml version="1.0" encoding="ISO-8859-1" ?>
- <DOCUMENT>
  - <ELEMENT>
      <gdp>2.585</gdp>
      <population>82,422,299</population>
      <gdpunit>trillion</gdpunit>
      <milexpendpercent>1.5</milexpendpercent>
    </ELEMENT>
  </DOCUMENT>
```

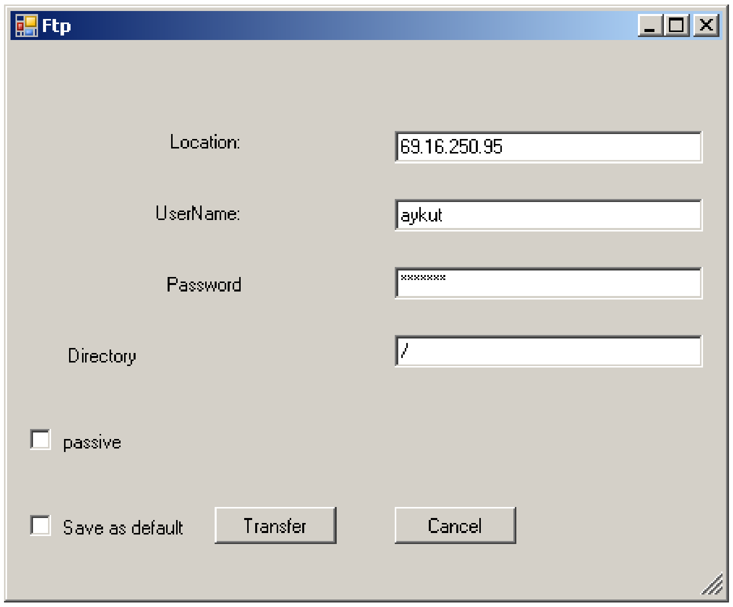
Step 11 Optional Web Service Creation

While you were going through spec file creation web service code was also created from your spec file. In order to put that web service into action, you need to save it first, and transfer the files into a server that supports .NET framework.

First save your web service by clicking on Web Service menu item and clicking on Save. Identify a location to save:

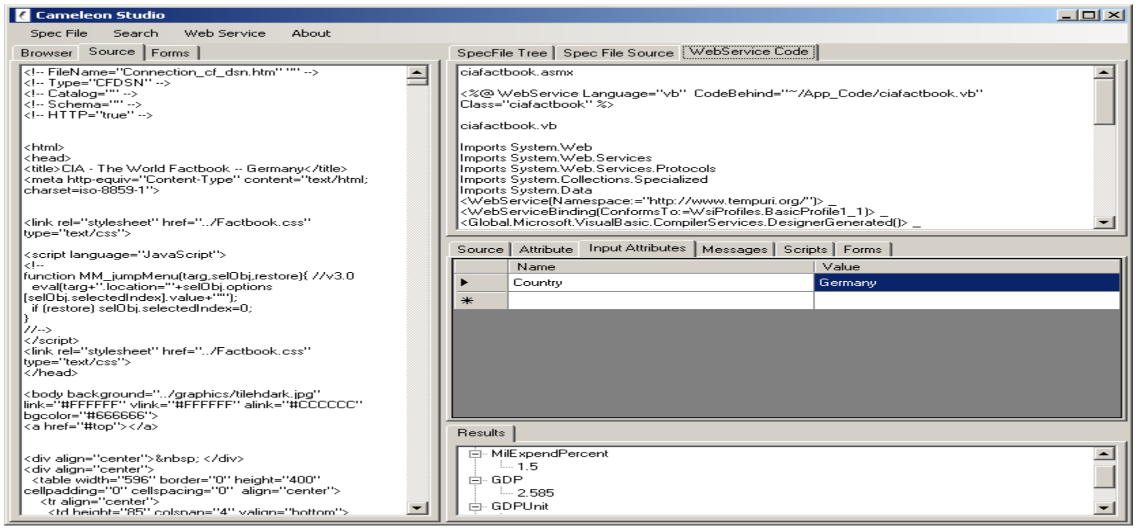


Now, click on the Web Service menu item and select Transfer: Again enter the required information for your transfer location.



Location:	69.16.250.95
UserName:	aykut
Password	*******
Directory	/
☐ passive	
☐ Save as default	

Note that there are two files that are being transferred. One of them is the asmx file that goes into the directory indicated in the ftp info above. You also need to create an App_Code directory in your server for the vb file as that is where the vb is assumed to exist as shown in the Web Service Code tab below:



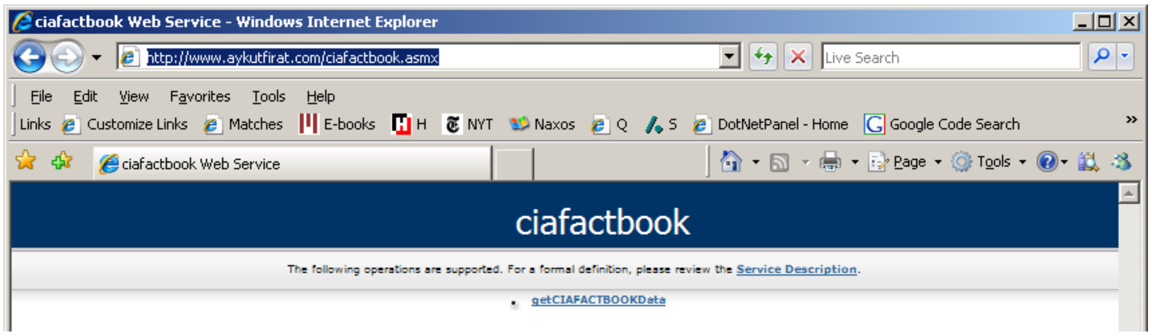
```
<!-- FileName="Connection_cf_dsn.htm" "" -->
<!-- Type="CFDSN" -->
<!-- Catalog="" -->
<!-- Schema="" -->
<!-- HTTP="true" -->


<html>
<head>
<title>CIA - The World Factbook -- Germany</title>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1">


<link rel="stylesheet" href="../Factbook.css"
type="text/css">

<script language="JavaScript">
<!--
function MM_jumpMenu(targ,selObj,restore){ //v3.0
  eval(targ+".location='"+selObj.options
[selObj.selectedIndex].value+"'");
  if (restore) selObj.selectedIndex=0;
}
//-->
</script>
<link rel="stylesheet" href="../Factbook.css"
type="text/css">
</head>

<body background="../graphics/tilehdark.jpg"
link="#FFFFFF" vlink="#FFFFFF" alink="#CCCCCC"
bgcolor="#666666">
<a href="#top"></a>


<div align="center">&nbsp; </div>
<div align="center">
  <table width="596" border="0" height="400"
cellpadding="0" cellspacing="0"  align="center">
    <tr align="center">
      <td height="85" colspan="4" valign="bottom">
```

```
ciafactbook.asmx

<%@ WebService Language="vb"  CodeBehind="~/App_Code/ciafactbook.vb"
Class="ciafactbook" %>

ciafactbook.vb

Imports System.Web
Imports System.Web.Services
Imports System.Web.Services.Protocols
Imports System.Collections.Specialized
Imports System.Data
<WebService(Namespace:="http://www.tempuri.org/")> _
<WebServiceBinding(ConformsTo:=WsiProfiles.BasicProfile1_1)> _
<Global.Microsoft.VisualBasic.CompilerServices.DesignerGenerated()> _
```

	Name	Value
▶	Country	Germany
*		

```
⊟ MilExpendPercent
    1.5
⊟ GDP
    2.585
⊟ GDPUnit
```

Once I do the transfer, I can go into my web service location (which is server name+specfilename.asmx, in my case this is: http://www.aykutfirat.com/ciafactbook.asmx



ciafactbook Web Service - Windows Internet Explorer

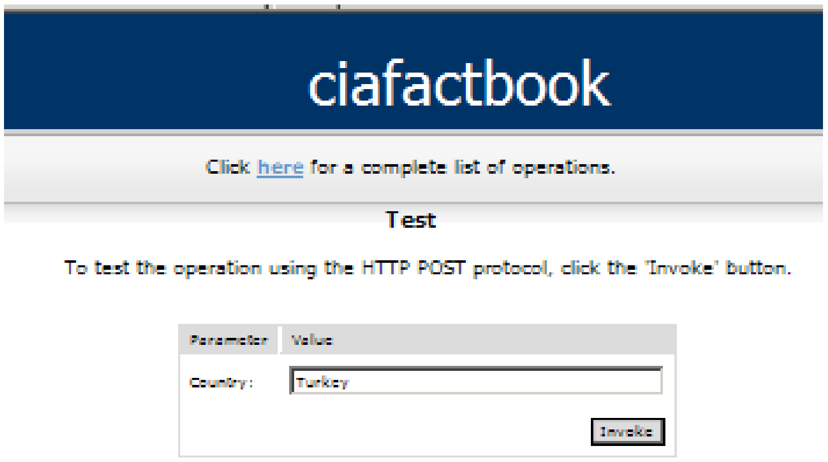
http://www.aykutfirat.com/ciafactbook.asmx

ciafactbook

The following operations are supported. For a formal definition, please review the Service Description.

- getCIAFACTBOOKData

As seen above this web service has a default method called getCIFACTBOOKData. Click on that, enter a country name, and click on invoke. (Note: You should enable remote testing in your server to test the method remotely.)



ciafactbook

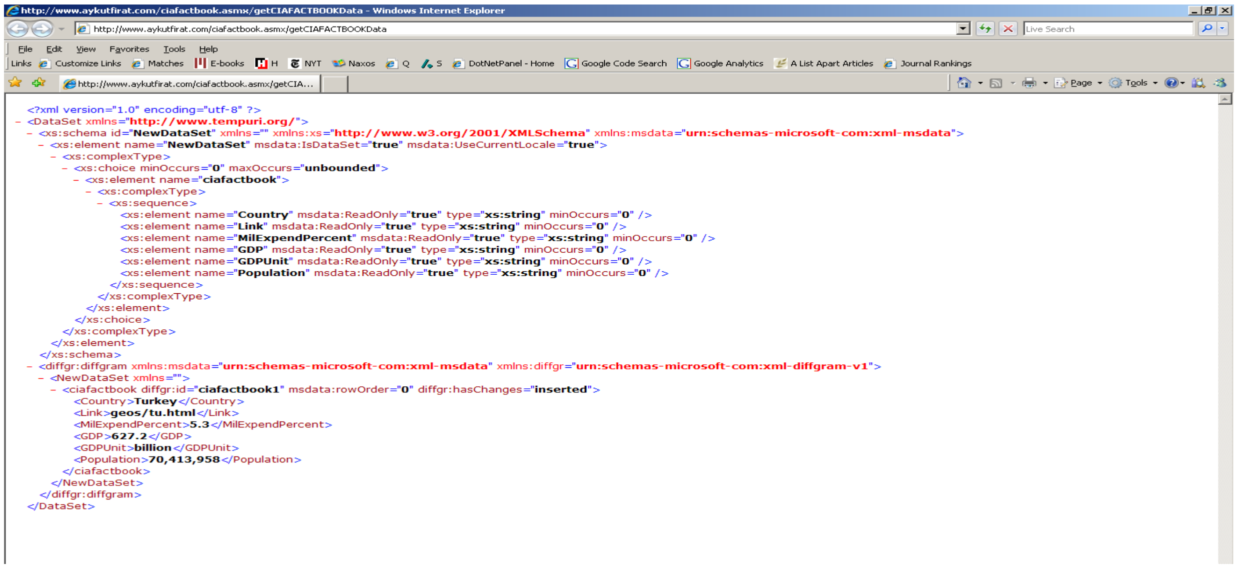
Click here for a complete list of operations.

Test

To test the operation using the HTTP POST protocol, click the 'Invoke' button.

Parameter	Value
Country:	Turkey

The returned results are in the form of XML, which takes the form of a data set in the .NET framework.



http://www.aykutfirat.com/ciafactbook.asmx/getCIAFACTBOOKData - Windows Internet Explorer

http://www.aykutfirat.com/ciafactbook.asmx/getCIAFACTBOOKData

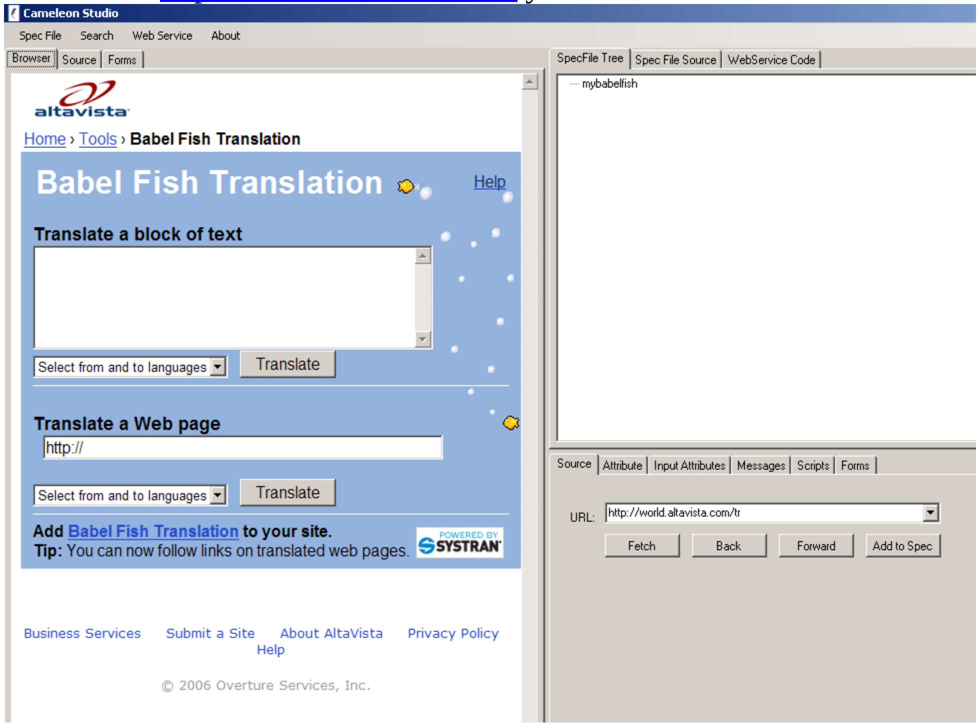
```
  <?xml version="1.0" encoding="utf-8" ?>
- <DataSet xmlns="http://www.tempuri.org/">
  - <xs:schema id="NewDataSet" xmlns="" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
    - <xs:element name="NewDataSet" msdata:IsDataSet="true" msdata:UseCurrentLocale="true">
      - <xs:complexType>
        - <xs:choice minOccurs="0" maxOccurs="unbounded">
          - <xs:element name="ciafactbook">
            - <xs:complexType>
              - <xs:sequence>
                  <xs:element name="Country" msdata:ReadOnly="true" type="xs:string" minOccurs="0" />
                  <xs:element name="Link" msdata:ReadOnly="true" type="xs:string" minOccurs="0" />
                  <xs:element name="MilExpendPercent" msdata:ReadOnly="true" type="xs:string" minOccurs="0" />
                  <xs:element name="GDP" msdata:ReadOnly="true" type="xs:string" minOccurs="0" />
                  <xs:element name="GDPUnit" msdata:ReadOnly="true" type="xs:string" minOccurs="0" />
                  <xs:element name="Population" msdata:ReadOnly="true" type="xs:string" minOccurs="0" />
                </xs:sequence>
              </xs:complexType>
            </xs:element>
          </xs:choice>
        </xs:complexType>
      </xs:element>
    </xs:schema>
  - <diffgr:diffgram xmlns:msdata="urn:schemas-microsoft-com:xml-msdata" xmlns:diffgr="urn:schemas-microsoft-com:xml-diffgram-v1">
    - <NewDataSet xmlns="">
      - <ciafactbook diffgr:id="ciafactbook1" msdata:rowOrder="0" diffgr:hasChanges="inserted">
          <Country>Turkey</Country>
          <Link>geos/tu.html</Link>
          <MilExpendPercent>5.3</MilExpendPercent>
          <GDP>627.2</GDP>
          <GDPUnit>billion</GDPUnit>
          <Population>70,413,958</Population>
        </ciafactbook>
      </NewDataSet>
    </diffgr:diffgram>
</DataSet>
```

1. Advanced Features of the Cameleon# Studio

Although the above example covers most of the features of the Cameleon# Studio, there are a couple of things that were not mentioned. These are explained below:

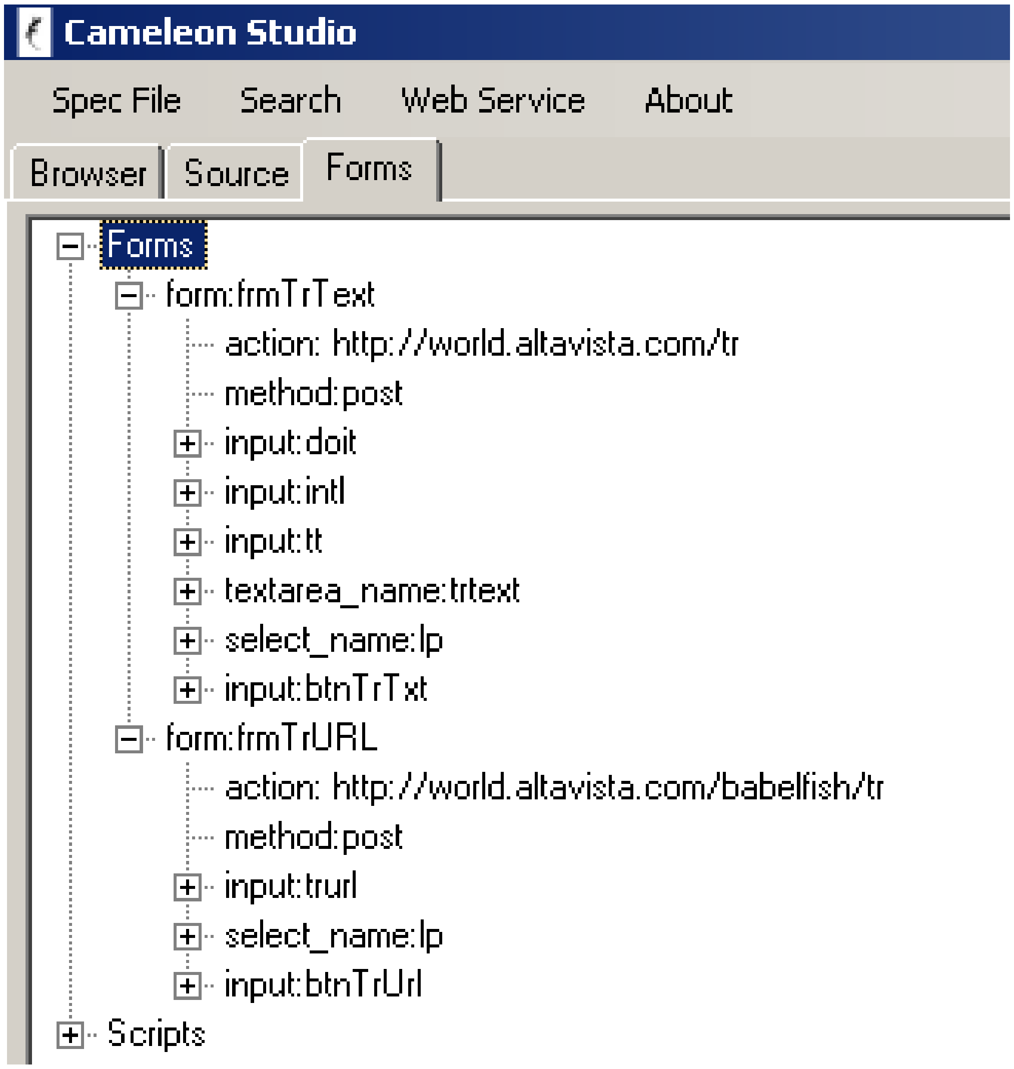
Auto form handling

If the web page you want to wrap requires form submission before you can access your data you need to have a form submission. For example if you want to wrap babelfish at http://world.altavista.com/tr you need to work with forms



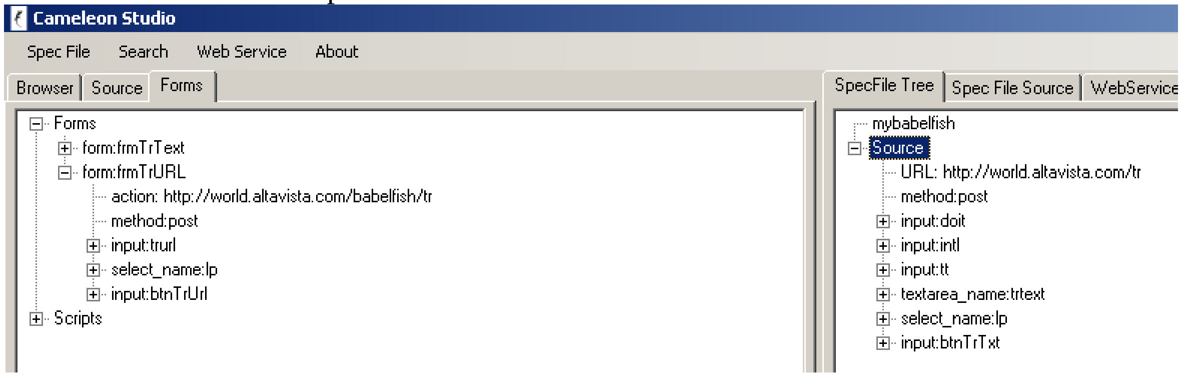
URL: http://world.altavista.com/tr

If you click on the Forms tab in the babelfish page, you will see two forms:



```
Forms
  form:frmTrText
    action: http://world.altavista.com/tr
    method:post
    input:doit
    input:intl
    input:tt
    textarea_name:trtext
    select_name:lp
    input:btnTrTxt
  form:frmTrURL
    action: http://world.altavista.com/babelfish/tr
    method:post
    input:trurl
    select_name:lp
    input:btnTrUrl
Scripts
```

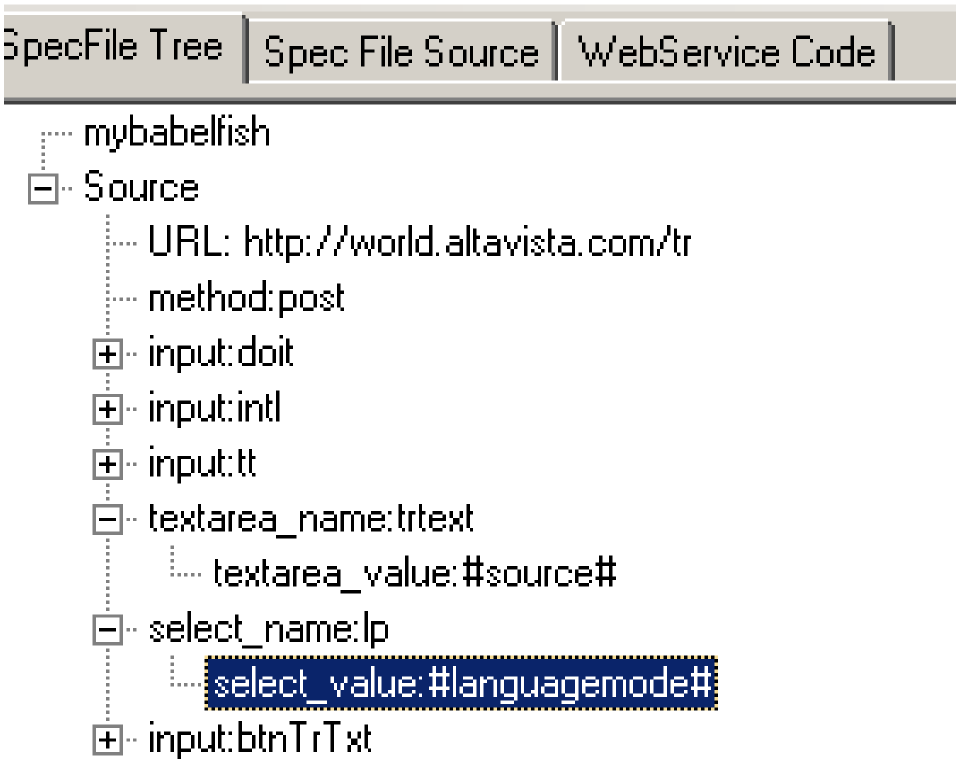
The first form is the one we are interested in. If you double click on form:frmTrText it will transferred to the specFile Tree tab as a source.



```
Forms
  form:frmTrText
  form:frmTrURL
    action: http://world.altavista.com/babelfish/tr
    method:post
    input:trurl
    select_name:lp
    input:btnTrUrl
Scripts
```

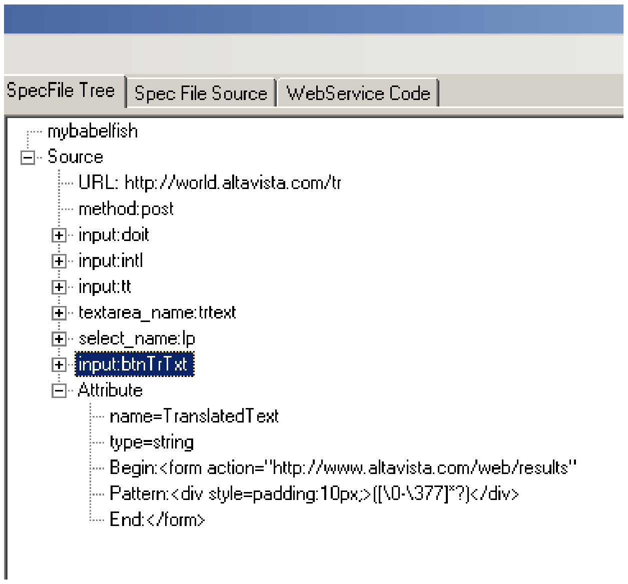
```
mybabelfish
Source
  URL: http://world.altavista.com/tr
  method:post
  input:doit
  input:intl
  input:tt
  textarea_name:trtext
  select_name:lp
  input:btnTrTxt
```

We then need to edit this form to encode the input attributes namely the text to be translated, and the mode of translation (i.e. from which language to which language). We will call these two input attributes as source and language mode.



```
mybabelfish
Source
    URL: http://world.altavista.com/tr
    method:post
    input:doit
    input:intl
    input:tt
    textarea_name:trtext
        textarea_value:#source#
    select_name:lp
        select_value:#languagemode#
    input:btnTrTxt
```

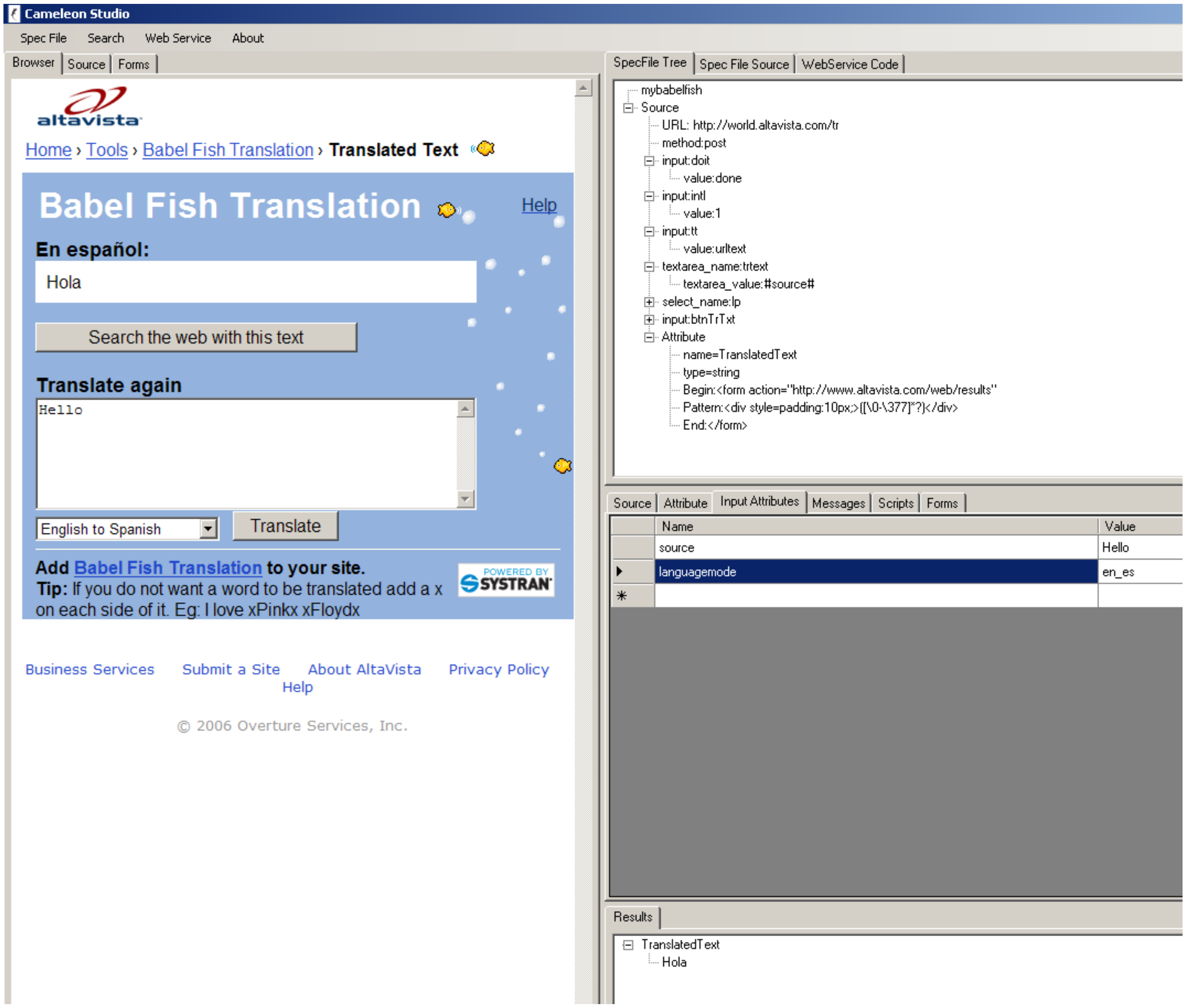
Then we add an attribute for extracting the results:



```
mybabelfish
Source
    URL: http://world.altavista.com/tr
    method:post
    input:doit
    input:intl
    input:tt
    textarea_name:trtext
    select_name:lp
    input:btnTrTxt
    Attribute
        name=TranslatedText
        type=string
        Begin:<form action="http://www.altavista.com/web/results"
        Pattern:<div style=padding:10px;>([\0-\377]*?)</div>
        End:</form>
```

And test with some Input Attribute values:

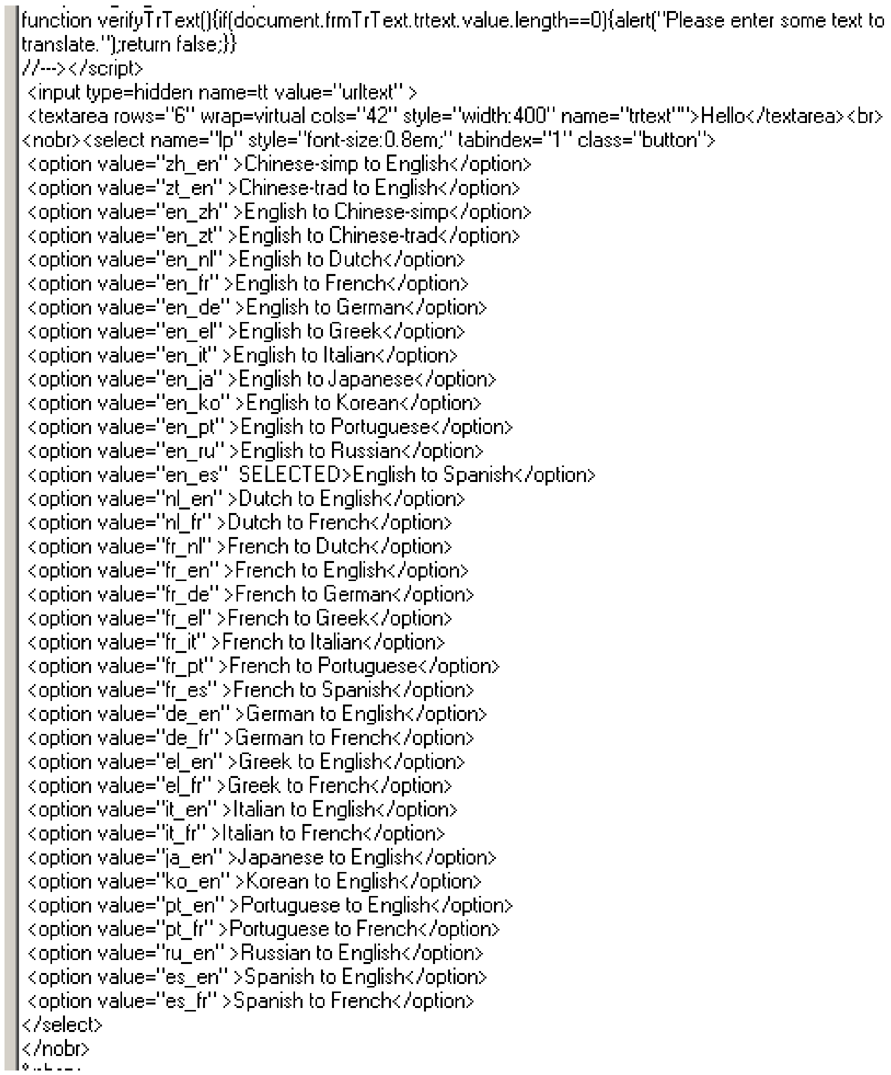
Source: Hello
Languagemode: en_es
Note that the language mode values should be taken from the acceptable values of babelfish by examining its source as shown below.



```
mybabelfish
Source
    URL: http://world.altavista.com/tr
    method:post
    input:doit
        value:done
    input:intl
        value:1
    input:tt
        value:urltext
    textarea_name:trtext
        textarea_value:#source#
    select_name:lp
    input:btnTrTxt
    Attribute
        name=TranslatedText
        type=string
        Begin:<form action="http://www.altavista.com/web/results"
        Pattern:<div style=padding:10px;>([\0-\377]*?)</div>
        End:</form>
```

Name	Value
source	Hello
languagemode	en_es

```
TranslatedText
    Hola
```

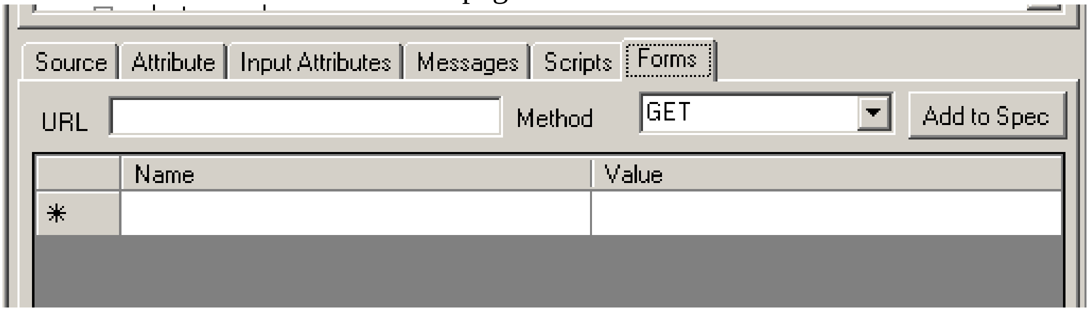


```
function verifyTrText(){if(document.frmTrText.trtext.value.length==0){alert("Please enter some text to
translate.");return false;}}
//--></script>
 <input type=hidden name=tt value="urltext" >
 <textarea rows="6" wrap=virtual cols="42" style="width:400" name="trtext"">Hello</textarea><br>
<nobr><select name="lp" style="font-size:0.8em;" tabindex="1" class="button">
 <option value="zh_en" >Chinese-simp to English</option>
 <option value="zt_en" >Chinese-trad to English</option>
 <option value="en_zh" >English to Chinese-simp</option>
 <option value="en_zt" >English to Chinese-trad</option>
 <option value="en_nl" >English to Dutch</option>
 <option value="en_fr" >English to French</option>
 <option value="en_de" >English to German</option>
 <option value="en_el" >English to Greek</option>
 <option value="en_it" >English to Italian</option>
 <option value="en_ja" >English to Japanese</option>
 <option value="en_ko" >English to Korean</option>
 <option value="en_pt" >English to Portuguese</option>
 <option value="en_ru" >English to Russian</option>
 <option value="en_es"  SELECTED>English to Spanish</option>
 <option value="nl_en" >Dutch to English</option>
 <option value="nl_fr" >Dutch to French</option>
 <option value="fr_nl" >French to Dutch</option>
 <option value="fr_en" >French to English</option>
 <option value="fr_de" >French to German</option>
 <option value="fr_el" >French to Greek</option>
 <option value="fr_it" >French to Italian</option>
 <option value="fr_pt" >French to Portuguese</option>
 <option value="fr_es" >French to Spanish</option>
 <option value="de_en" >German to English</option>
 <option value="de_fr" >German to French</option>
 <option value="el_en" >Greek to English</option>
 <option value="el_fr" >Greek to French</option>
 <option value="it_en" >Italian to English</option>
 <option value="it_fr" >Italian to French</option>
 <option value="ja_en" >Japanese to English</option>
 <option value="ko_en" >Korean to English</option>
 <option value="pt_en" >Portuguese to English</option>
 <option value="pt_fr" >Portuguese to French</option>
 <option value="ru_en" >Russian to English</option>
 <option value="es_en" >Spanish to English</option>
 <option value="es_fr" >Spanish to French</option>
</select>
</nobr>
```

Then save and transfer your spec file to complete the task.

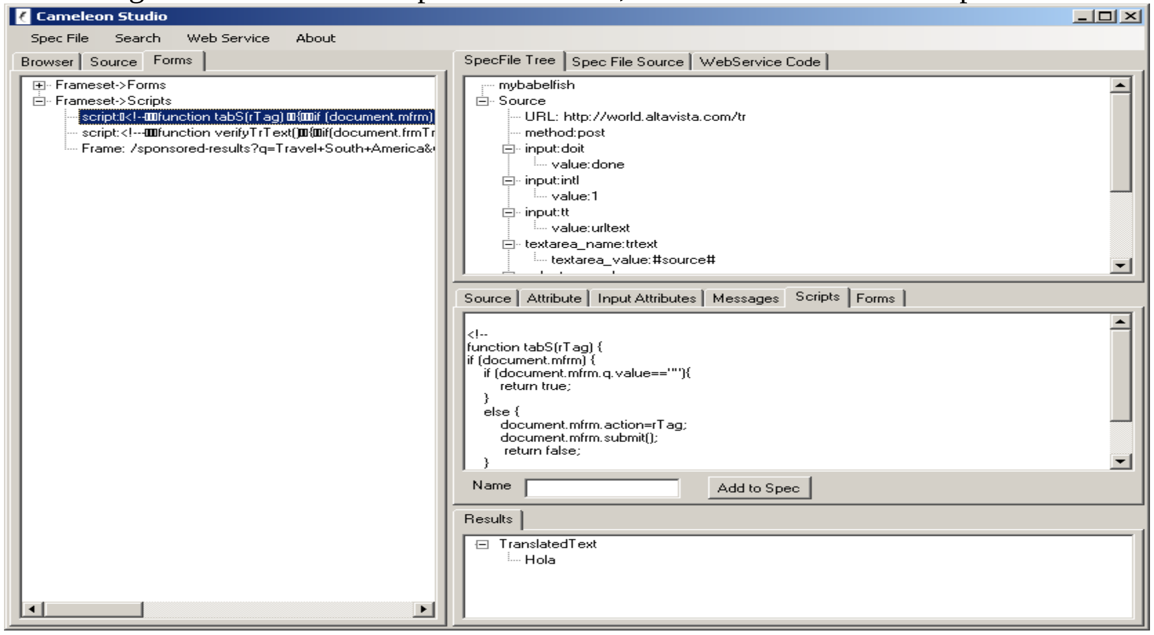
Custom Forms

If you want to use custom forms you can define them in the Forms tab by entering its URL, method, and name value pairs and clicking on Add to Spec. This is valuable when you cannot do the transfer as in the previous example, or your form is somehow different than what is shown on the page.



Scripts

This feature is not used as frequently it may sometimes be needed. Scripts in web pages are shown in the left-most Forms tab and can be transferred to the Scripts tab with a single click. Here the script can be edited, named and added to the spec file.



```
Frameset->Forms
Frameset->Scripts
    script:<!--function tabS(rTag) {if (document.mfrm)
    script:<!--function verifyTrText(){if(document.frmTr
    Frame: /sponsored-results?q=Travel+South+America&
```

```
mybabelfish
Source
    URL: http://world.altavista.com/tr
    method:post
    input:doit
        value:done
    input:intl
        value:1
    input:tt
        value:urltext
    textarea_name:trtext
        textarea_value:#source#
```

```
<!--
function tabS(rTag) {
if (document.mfrm) {
   if (document.mfrm.q.value==""){
      return true;
   }
   else {
      document.mfrm.action=rTag;
      document.mfrm.submit();
       return false;
   }
```

```
TranslatedText
    Hola
```

An example spec file that relies on scripts is Expedia. As shown below Expedia has a JSCRIPT tag called Time which refers to the result of interpreting some JavaScript code. In such a case the script can be modified in the scripts window, and added to the spec file.

```
<SOURCE URI="http://www.expedia.com/pub/agent.dll">
        <COOKIE name="path"><![CDATA[/]]></COOKIE>
        <JSCRIPT name="Time"><![CDATA[var d; d = new Date(); print(d.getTime());]]></JSCRIPT>
        <POST method="GET">
                <PARAM name="qscr" value="fexp"/>
                <PARAM name="flag" value="q"/>
                <PARAM name="city1" value="#Departure#"/>
                <PARAM name="citd1" value="#Destination#" />
                <PARAM name="date1" value="#Date1#" />
                <PARAM name="time1" value="362"/>
                <PARAM name="date2" value="#Date2#"/>
                <PARAM name="time2" value="362"/>
                <PARAM name="cAdu" value="1"/>
                <PARAM name="rfrr" value="-429"/>
                <PARAM name="zz" value="#Time#"/>

        </POST>
```

Other Features

You can open existing spec files with the Spec File open directive, view error messages in the Messages pane.

2. Known Bugs

Cameleon# Studio is an experimental system and contains a number of known bugs. Some of these are:

1) Files transferred message does not always mean that files are actually transferred. If the file was not found in the directory it would give the message although the file was not transferred.

2) You need to always start with the SpecFile new directive and give a name to your spec file. If you forget this spec file transfer will not work. You can still manually transfer your file though.

3) Cameleon# Studio and Cameleon# Server will produce different results in some rare cases. This is because the simulation performed in Cameleon# Studio is slightly different in some cases.

There are also some other bugs that you may find. Please keep in mind that this is a pre-beta version, and the software is continuously being updated.

Manually Authoring Cameleon# Spec Files

The general layout of a spec file is first presented in this guide, followed by a closer look at some of the useful features. A summary of the spec file syntax is provided at the end.

Cameleon# Spec File Structure

Cameleon# spec file is XML-based, which means that its content is organized as a hierarchy of nodes, or tags. Always name the spec file with xml extension, e.g., my_spec_file.xml. Let's examine the sample spec file below to illustrate spec file structure, as well as some key elements of a spec file.

```
1  <?xml version='1.0' encoding="UTF-8"?>
2  <!DOCTYPE RELATION SYSTEM "http://interchange.mit.edu/cameleon_sharp/cameleonspec.dtd">
3  <!--Comments: This is a spec file to wrap the CIA Fact Book site.-->
4  <RELATION name="cia">
5     <SOURCE URI="http://www.odci.gov/cia/publications/factbook/index.html">
6        <ATTRIBUTE name="Link" type="String">
7            <BEGIN><![CDATA[<body]]></BEGIN>
8            <PATTERN><![CDATA[<option value="([^"]*)"[^>]*>#Country#]]></PATTERN>
9            <END><![CDATA[</[Bb][oO][dD][yY]>]]></END>
10       </ATTRIBUTE>
11       …
12       <ATTRIBUTE…>…</ATTRIBUTE>
13    </SOURCE>
14    …
15    <SOURCE…>…</SOURCE>
16 </RELATION>
```

Line 1: Standard XML declaration tag. Every XML document, hence every spec file, starts with some kind of declaration similar to this. You can copy this line and use it in your spec file as is.

Line 2: DTD schema tag. DTD schema is a way to validate a XML document. The schema, located at http://interchange.mit.edu/cameleon_sharp/cameleonspec.dtd, contains information about the tags and their signatures[3]. You can copy this second line and use it in your spec file as is. Below is the cameleonspec.dtd

```
<!ELEMENT RELATION (SOURCE+)>
<!ATTLIST RELATION name CDATA #REQUIRED>
<!ELEMENT SOURCE (AUTHENTICATION?, COOKIE*, JSCRIPT?, POST?, ATTRIBUTE*)>
<!ATTLIST SOURCE URI CDATA #REQUIRED>
<!ATTLIST SOURCE DELAY CDATA #IMPLIED>
```

[3] Sometimes it may be a good idea to validate your spec file against the DTD schema for debugging purpose. Here is a reliable XML validator to do that: http://www.cogsci.ed.ac.uk/~richard/xml-check.html.

```
<!ELEMENT AUTHENTICATION (Realm, Username, Password)>
<!ELEMENT Realm (#PCDATA)>
<!ELEMENT Username (#PCDATA)>
<!ELEMENT Password (#PCDATA)>
<!ELEMENT COOKIE (#PCDATA)>
<!ATTLIST COOKIE name CDATA #REQUIRED>
<!ELEMENT JSCRIPT (#PCDATA)>
<!ATTLIST JSCRIPT name CDATA #REQUIRED>
<!ELEMENT POST (PARAM+)>
<!ATTLIST POST method (POST|GET) "POST">
<!ELEMENT PARAM (#PCDATA)>
<!ATTLIST PARAM
        name CDATA #REQUIRED
        value CDATA #REQUIRED>
<!ELEMENT ATTRIBUTE (PREFIX?, SUFFIX?, BEGIN, OBJECT?, PATTERN+,
END)>
<!ELEMENT PREFIX (#PCDATA)>
<!ELEMENT SUFFIX (#PCDATA)>
<!ATTLIST ATTRIBUTE
        name CDATA #REQUIRED
        type (String) "String">
<!ELEMENT BEGIN (#PCDATA)>
<!ELEMENT OBJECT (OBJECT_BEGIN, OBJECT_END)>
<!ELEMENT OBJECT_BEGIN (#PCDATA)>
<!ELEMENT OBJECT_END (#PCDATA)>
<!ELEMENT PATTERN (#PCDATA)>
<!ELEMENT END (#PCDATA)>
```

Line 3: Comment tag. Comments in an XML file are enclosed by the <!-- *your comments* --> tag. You can intersperse comments anywhere you like in your spec file, as long as the well-formedness of the XML document is observed. Anything within the comment tag is ignored by the Cameleon# engine.

Line 4: RELATION tag. Relation is the root element of a spec file. It contains one or more SOURCE tags, which we will look at next. It can be thought of as the declaration of a relation, or table, in a database. Unlike traditional database relation, a Cameleon# relation can consist of attributes from multiple sources (much like a database view). Although the name attribute of this tag concurrently has no function, a meaningful name should be assigned to it for readability. A good convention is to use the same name as the name of the spec file (i.e. the relation *cia* is derived from *cia.xml*).

Line 5: SOURCE tag. Each source tag signifies a single web resource. While static web pages are obvious web resources, Java servlets, Perl script cgi, or any server side programs that return data over the web can be web resources as well. The URI attribute of the source tag specifies the location of a web resource. Use http encoding for special characters in a URI, e.g., use *&amp;* for *&*. The content of any specified web resource will be retrieved for pattern matching to form attributes, as we will see in the discussion of attribute tag below. Each source tag can contain multiple attribute tags.

Line 6: ATTRIBUTE tag. An attribute tag defines what is to be extracted from a specified source. Pattern matching by regular expression is used to locate the information to be extracted (see discussion of PATTERN tag below). Each attribute must be assigned a unique name and a type. Currently, all data extracted are treated as string data type by Cameleon#. So, you can set type="String" for every attribute you define. Once an attribute is defined, it can be referenced by a user issued query (i.e. via the Cameleon# GUI), or it can be referenced internally in the spec file. The latter usage is similar to declaring a variable to store information extracted from one source for subsequent use within the spec file. For example, it is a common scenario to extract a URL address from one source and use that address as the URI of the next source element. When referencing an attribute internally, simply put # symbols around the attribute name (i.e., #attribute_name#). Attributes can be referenced anywhere in a spec file where string value is expected. They can even be embedded within plain static text, such as in a matching pattern. In these cases, an extra attribute *link="true"* for the tag is needed.

Line 7: BEGIN tag. The begin tag contains a regular expression that matches the beginning of a region of text that contains desired data to be extracted. This helps to limit the scope of matching, and thus improves the efficiency in finding patterns. A special XML tag <![CDATA[*regular_expression*]]> is used to encase the regular expression. This tag ensures that any characters in your regular expression that potentially invalidates the entire XML document are ignored. Remember, every spec file is parsed by a XML parser before processed. The CDATA tag tells the XML parser not to parse the stuff inside it. Therefore, always surround your regular expression with a CDATA tag.

Line 8: PATTERN tag. The pattern tag pinpoints the location of the desired piece of data to be extracted. Similar to the begin tag, the content of the pattern tag is a regular expression (surrounded by the CDATA tag). The scope of matching is only limited to the region set by BEGIN and END tags. Notice the parenthesis "( )" within the regular expression. The parenthesis is there to indicate the segment of characters to extract, provided the surrounding patterns match. That segment of extracted characters is set to be the value of the particular attribute. The pattern is matched within the BEGIN/END region as many times as possible. Therefore, an attribute can sometimes have multiple values. When the result of a query involves two or more attributes, the Cartesian product between the individual values of each attribute will be returned. The only time a Cartesian product is not taken is when the number of values across all involved attributes are the same. In that case, the values are merged instead. For instance, the i^{th} value of attribute *a* will be merged with the i^{th} value of attribute *b*, and so on, to produce the i^{th} row of result.

Line 9: END tag. The end tag is analogous to the begin tag. It marks the end of a region where pattern matching should be applied for a particular attribute.

Line 10: Signifies the end of an attribute declaration.

Line 11-12: Signify the possibility of more than one attribute tag per source.
Line 13: Signifies the end of a source declaration.
Line 14-15: Signify the possibility of more than one source tag per relation.
Line 16: Signifies the end of the relation declaration.

Features of Spec Files

Disjunctive Patterns

Sometimes it is not possible to discover a single pattern that would match the desired data across all similar pages. In these types of cases we allow disjunctive patterns to specify multiple patterns. The following is an example of such a situation in which two disjunctive patterns are specified for a single attribute.

```
<ATTRIBUTE name="LastTrade " type="String">
<BEGIN><![CDATA[Last\s*Trade]]></BEGIN>
<END><![CDATA[</TR>]]></END>
<PATTERN><![CDATA[<B>\s*(.*?)\s*<FONT\s*SIZE=1>(.*?)</FONT>]]></PATTERN>
<PATTERN><![CDATA[<B>\s*(\d+)\s*</B>]]></PATTERN>
</ATTRIBUTE>
```

We should note, however, that these disjunctive patterns are not mutually exclusive and occasionally special care must be taken to construct patterns whose intersections are empty. Otherwise the same item will be matched multiple times and repeated in the output.

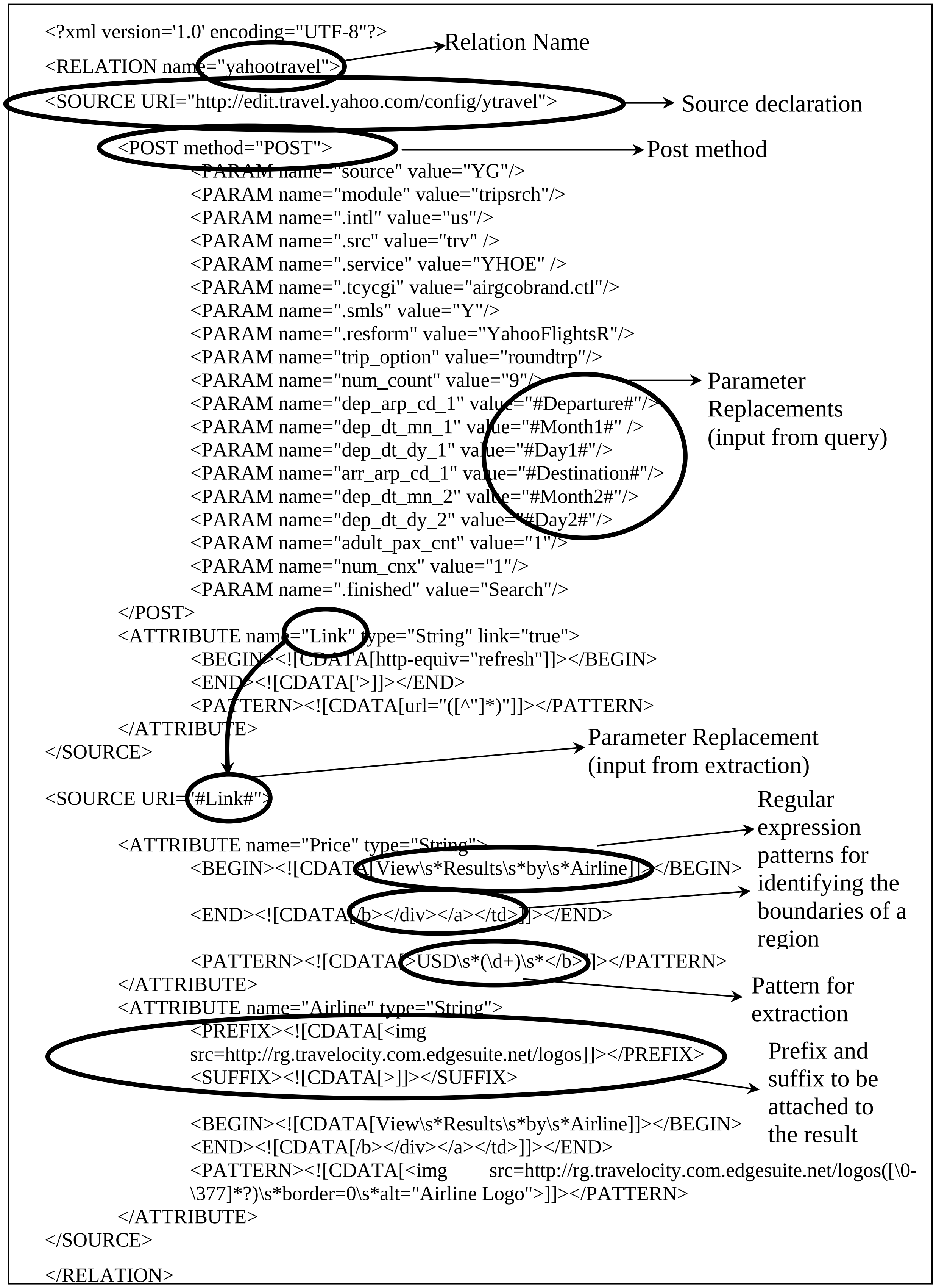
Conjunction

In spec files it is possible to define conjunctive patterns, by simply denoting them with enclosing parentheses. The semantics of conjunctive patterns in Cameleon# corresponds to the concatenation of pattern matches. The first pattern element in the example above has such a case with two groups of enclosing parentheses, the first one matching the whole part of last trade value, the second one the fractional part. The matched elements are then concatenated to form a single value. This feature is very useful, when data to be extracted is not atomic, and separated by unwanted tags.

Multi-page transitions

When wrapping web pages, we sometimes need to traverse multiple pages to locate the page we want to extract information from. This situation occurs when the URLs are created dynamically (e.g. a session ID is assigned for each access to the page), or a cookie needs to be established before you can go to the desired page, or there is no simple way of deducing the desired URL without visiting a particular page, or the data is spread through multiple pages. To handle these kinds of cases Cameleon# has a feature that lets us wrap multiple pages for a relation.

In Figure 1 for example the link to the second page is extracted from the first web page. We need to perform this step in this case because there is no simple way of deducing in advance what the link is supposed to be. Then the link is supplied to the next source element, which takes us to the page where we want to extract the values of price and airline.



```
<?xml version='1.0' encoding="UTF-8"?>
<RELATION name="yahootravel">
<SOURCE URI="http://edit.travel.yahoo.com/config/ytravel">
        <POST method="POST">
                <PARAM name="source" value="YG"/>
                <PARAM name="module" value="tripsrch"/>
                <PARAM name=".intl" value="us"/>
                <PARAM name=".src" value="trv" />
                <PARAM name=".service" value="YHOE" />
                <PARAM name=".tcycgi" value="airgcobrand.ctl"/>
                <PARAM name=".smls" value="Y"/>
                <PARAM name=".resform" value="YahooFlightsR"/>
                <PARAM name="trip_option" value="roundtrp"/>
                <PARAM name="num_count" value="9"/>
                <PARAM name="dep_arp_cd_1" value="#Departure#"/>
                <PARAM name="dep_dt_mn_1" value="#Month1#" />
                <PARAM name="dep_dt_dy_1" value="#Day1#"/>
                <PARAM name="arr_arp_cd_1" value="#Destination#"/>
                <PARAM name="dep_dt_mn_2" value="#Month2#"/>
                <PARAM name="dep_dt_dy_2" value="#Day2#"/>
                <PARAM name="adult_pax_cnt" value="1"/>
                <PARAM name="num_cnx" value="1"/>
                <PARAM name=".finished" value="Search"/>
        </POST>
        <ATTRIBUTE name="Link" type="String" link="true">
                <BEGIN><![CDATA[http-equiv="refresh"]]></BEGIN>
                <END><![CDATA['>]]></END>
                <PATTERN><![CDATA[url="([^"]*)"]]></PATTERN>
        </ATTRIBUTE>
</SOURCE>

<SOURCE URI="#Link#">

        <ATTRIBUTE name="Price" type="String">
                <BEGIN><![CDATA[View\s*Results\s*by\s*Airline]]></BEGIN>

                <END><![CDATA[/b></div></a></td>]]></END>

                <PATTERN><![CDATA[>USD\s*(\d+)\s*</b>]]></PATTERN>
        </ATTRIBUTE>
        <ATTRIBUTE name="Airline" type="String">
                <PREFIX><![CDATA[<img
                src=http://rg.travelocity.com.edgesuite.net/logos]]></PREFIX>
                <SUFFIX><![CDATA[>]]></SUFFIX>

                <BEGIN><![CDATA[View\s*Results\s*by\s*Airline]]></BEGIN>
                <END><![CDATA[/b></div></a></td>]]></END>
                <PATTERN><![CDATA[<img        src=http://rg.travelocity.com.edgesuite.net/logos([\0-
                \377]*?)\s*border=0\s*alt="Airline Logo">]]></PATTERN>
        </ATTRIBUTE>
</SOURCE>

</RELATION>
```

Relation Name
Source declaration
Post method
Parameter Replacements (input from query)
Parameter Replacement (input from extraction)
Regular expression patterns for identifying the boundaries of a region
Pattern for extraction
Prefix and suffix to be attached to the result

Figure 1 Cameleon Spec-File for Yahoo Travel

Parameter Replacement

Parameter replacement is the use of input or extracted attribute values within the subsequent elements in the spec file. In the multiple page traversal case, we have seen one example of this. The value of attribute “Link” was used in the next source element. It is also possible to supply any extracted or input attribute value within the attribute definitions. Consider the following SQL query to the Yahoo Travel Web site:

Select Airline, Price from yahootravel where Departure=“BOS” and Destination=“SFO” and Month1=“5” and Day1= “19” and Month2= “6” and Day2= “1”

When this query is executed, the input attribute values specified after the *where* clause replace the same name attributes enclosed between # signs in the post parameters as shown in Figure 1.

Get, Post Methods & Authentication

Cameleon# spec files support both *get* and *post* methods when connecting to Web pages. A post example is shown in Figure 1. *Method* attribute of the post tag determines which method is to be used.

Most web pages perform authentication through forms. In connecting to those Web pages, get or post methods with parameter replacement can be used for authentication purposes. In some other pages, however, the authentication is done through pop-up password windows. We handle these kinds of cases with the following scheme: (The username and password values have to be inputted within the SQL query, since they are coded as references in this spec file.)

```
<SOURCE URI=" http://game.etrade.com/cgi-bin/cgitrade/TransHistory ">
<AUTHENTICATION>
   <Realm>E*Trade Player (game)</Realm>
   <Username>#username#</Username>
   <Password>#password#</Password>
</AUTHENTICATION>
```

Custom Cookies

In Cameleon# cookie handling is automatic as long as the cookies are set through headers. In some cases, cookies can be set in a non-standard way for example using Javascript API. To handle these cases we allow custom cookie setting as shown in example below (custom cookies used are: *jscript=1;path=/*)

```
   <SOURCE URI="http://www.expedia.com/pub/agent.dll">
   <COOKIE name="jscript">1</COOKIE>
   <COOKIE name="path">/</COOKIE>
```

JavaScript Interpretation

JavaScript is used frequently in Web pages in creating the html document on the client side. In most cases, JavaScript does not pose a problem in wrapping Web pages, because it is usually used for cosmetic reasons. In some cases, however, not being able to interpret JavaScript may block the wrapper engine in getting to a desired page. One real example is the Expedia Web site, which requires interpreting JavaScript code and supplying the result as a post parameter. Cameleon# spec files allow the interpretation of JavaScripts as shown in the following example.

```
   <SOURCE URI="http://www.expedia.com/pub/agent.dll">
   <JSCRIPT name="Time">
   var d; d = new Date(); print(d.getTime());
```

```
</JSCRIPT>
```

In this example, the output of the JavaScript snippet is assigned to the Time attribute. We should note that the JavaScript code to be interpreted does not have to be static, and parameter replacement can be used in JScript tags as well.

Prefix and Suffixes

Figure 1 shows an example of prefix and suffix declarations. With these constructs it is possible to add static text before and after the extracted data values. By using parameter replacement feature, it also becomes possible to glue multiple extractions together.

Delays

We should mention another useful feature in spec file creation: delays. This is used when the wrapper engine requests data from a Web site, but has to wait a certain amount of time before getting an answer. We cover this case by specifying a delay element in the source declarations specifying the waiting time in terms of milliseconds. We show an example below:

```
<SOURCE URI="http://www.qixo.com/#Link#" DELAY="65000">
```

Debugging

You can test your spec files by setting the *debug* parameter to *true* in the web-based query testing harness (http://interchange.mit.edu/cameleon_sharp/cameleon.html). It will output all web data retrieved by Cameloen, the sections of source data defined by your BEGIN/END tags of each attribute, the data extracted according to your pattern specification when your spec file works. Examine your extraction rule if the data you want to extract has been retrieved but you are not extracting them. Examine source and parameter specifications if Cameleon# cannot even retrieve the page content. Study the error messages, if any.

Summary of Cameleon# Spec File Syntax

Element	**Signature**	**Required**	**Parent**	**Attribute**	**Comment**
RELATION	<RELATION name="...">	1	none	name (required)	Root element.
SOURCE	<SOURCE URI="..." DELAY="...">	1 or more	RELATION	URI (required)	Source specifies a single web resource.
				DELAY	Delay by the specified number of milliseconds.
AUTHENTICATION	<AUTHENTICATION>	0 or 1	SOURCE		For websites that authentication through pop-up window.
Realm	<Realm>	1	AUTHENTICATION		Specifies an authentication realm.
Username	<Username>	1	AUTHENTICATION		Username for authentication.
Password	<Password>	1	AUTHENTICATION		Password for authentication.
COOKIE	<COOKIE name="...">	0 or more	SOURCE	name (required)	Each cookie has a name.
JSCRIPT	<JSCRIPT name="…">	0 or more	SOURCE	name (required)	Evaluates a JavaScript snippet and store output.
POST	<POST method="POST|GET">	0 or 1	SOURCE	method	Default value is "POST". Specifies the type of http request operation.
PARAM	<PARAM name="…" value="…">	1 or more	POST	name (required)	Input parameter to http request.
				value (required)	Value of input parameter.
ATTRIBUTE	<ATTRIBUTE name="…" type=""...">	0 or more	SOURCE	name (required)	Stores extracted values.
				type	Default is "String". Only string type is allowed.
PREFIX	<PREFIX>	0 or 1	ATTRIBUTE		Adds text to beginning of extracted value
SUFFIX	<SUFFIX>	0 or 1	ATTRIBUTE		Adds text to end of extracted value
BEGIN	<BEGIN>	1	ATTRIBUTE		Pattern for beginning of search region.
END	<END>	1	ATTRIBUTE		Pattern for end of search region.
OBJECT	<OBJECT>	0 or more	ATTRIBUTE		Defines object region that constitutes a tuple.
OBJECT_BEGIN	<OBJECT_BEGIN>	1	OBJECT		Pattern for beginning of object region.
OBJECT_END	<OBJECT_END>	1	OBJECT		Pattern for end of object region.
PATTERN	<PATTERN>	1 or more	ATTRIBUTE		Pattern that contains the data to be extracted.

Appendices

A1. Related Readings and References

[1] Aykut Firat, Stuart Madnick, Michael Siegel, "The Cameleon Web Wrapper Engine," *Proceedings of the Workshop on Technologies for E-Services*, September 14-15, 2000, Cairo, Egypt, http://web.mit.edu/smadnick/www/wp/2000-03.pdf

[2] Aykut Firat, Denis Peleshchuk, Prakash Rao, "iWrap: Instant Web Wrapper Generator," http://web.mit.edu/smadnick/www/wp/2000-10.pdf

[3] Tarik Alatovic (Thesis) "Capabilities Aware Planner/Optimizer/Executioner for COntext INterchange Project," http://web.mit.edu/smadnick/www/wp/2002-01.pdf

[4] Aykut Firat (PhD Thesis) "Information Integration Using Contextual Knowledge and Ontology Merging," Massachusetts Institute of Technology, Sloan School of Management, August, 2003.

[5] Shin Wee Chuang (Thesis), "A Taxonomy and Analysis of Web Wrapping Technologies," http://web.mit.edu/smadnick/www/wp/2004-08.pdf

[6] Aykut Firat, Stuart Madnick, Nor Adnan Yahaya, Choo Wai Kuan, Stéphane Bressan, "Information Aggregation using the Caméléon# Web Wrapper," (EC-Web) http://web.mit.edu/smadnick/www/wp/2005-06.pdf

[7] Lynn Wu, Aykut Firat, Tarik Alatovic, Stuart Madnick, "Querying Web-Sources within a Data Federation," (ICIS) http://web.mit.edu/smadnick/www/wp/2006-09.pdf

A.2 Regular Expressions

Syntax

What is a regular expression?
Perl5 regular expressions

It is beyond the scope of this guide to give a detailed explanation of regular expressions to beginners. The OROMatcher TM package is geared toward programmers who are already familiar with regular expressions, having used them with other languages, and who now want to apply them in their Java programs. However, we shall make a small attempt to cover the basics and summarize the Perl5 syntax supported by the OROMatcher TM Perl5 classes. For a detailed exploration of regular expressions for both beginners and advanced users, we recommend the book *Mastering Regular Expressions* by Jeffrey Friedl published by O'Reilly & Associates.

What is a regular expression?

Part of this discussion is based on page 94 of "Compilers, Principles, Techniques, and Tools" by Aho, Sethi and Ullman

A regular expression is a pattern denoted by a sequence of symbols representing a state-machine or mini-program that is capable of matching particular sequences of characters. Regular expressions have their root in lexical analysis and tokenization where a set of lexemes had to be recognized before being passed on to a parser. Since then, regular expressions took a life of their own, appearing in such languages as AWK, TCL, and of course Perl, for all sorts of textual data extraction and manipulation purposes.

The most basic regular expression syntax consists of 4 operations. Let A and B each represent an alphabet (a set of characters) and s and t represent members of those alphabets.

Operation	Representation	Meaning
Union of A and B	A|B	s is such that s is in A or s is in B
Concatentation of A and B	AB	st are such that s is in A and t is in B
Kleene closure of A	A*	Zero or more concatenations of A
Positive closure of A	A+	One or more concatenations of A

Using this notation you can define a regular expression for positive integers as follows:

```
digit +
```

Here digit represents the set of characters 0 - 9. A range of characters like this can be represented in most regular expression languages as `[0-9]`. Because this is such a common expression, some languages have a special character for it: `\d` .

Learning a regular expression language is quite simple once you've learned one, because most of the operations are the same. Only the notation changes.

Perl5 regular expressions

Here we summarize the syntax of Perl5 regular expressions, all of which is supported by the OROMatcher TM Perl5 classes. However, for a definitive reference, you should consult the `perlre` man page that accompanies the Perl5 distribution and also the book *Programming Perl, 2nd Edition* from O'Reilly & Associates. We need to point out here that for efficiency reasons the character set operator [...] is limited to work on only ASCII characters (Unicode characters 0 through 255). Other than that restriction, all Unicode characters should be useable in the package's regular expressions.

- Alternatives separated by |
- Quantified atoms

  {n,m} : Match at least n but not more than m times.
  {n,} : Match at least n times.
  {n} : Match exactly n times.
  * : Match 0 or more times.
  + : Match 1 or more times.
  ? : Match 0 or 1 times.

- Atoms
  - regular expression within parentheses
  - a . matches everything except \n
  - a ^ is a null token matching the beginning of a string or line (i.e., the position right after a newline or right before the beginning of a string)
  - a $ is a null token matching the end of a string or line (i.e., the position right before a newline or right after the end of a string)
  - Character classes (e.g., [abcd]) and ranges (e.g. [a-z])
    - Special backslashed characters work within a character class (except for backreferences and boundaries).
    - \b is backspace inside a character class
  - Special backslashed characters

    \b : null token matching a word boundary (\w on one side and \W on the other)
    \B : null token matching a boundary that isn't a word boundary
    \A : Match only at beginning of string
    \Z : Match only at end of string (or before newline at the end)
    \n : newline
    \r : carriage return
    \t : tab
    \f : formfeed
    \d : digit [0-9]
    \D : non-digit [^0-9]
    \w : word character [0-9a-z_A-Z]
    \W : a non-word character [^0-9a-z_A-Z]
    \s : a whitespace character [\t\n\r\f]
    \S : a non-whitespace character [^ \t\n\r\f]
    \xnn : hexadecimal representation of character

\cD : matches the corresponding control character
\nn or \nnn : octal representation of character unless a backreference.
\1, \2, \3, etc. : match whatever the first, second, third, etc. parenthesized group matched. This is called a backreference. If there is no corresponding group, the number is interpreted as an octal representation of a character.
\0 : matches null character
Any other backslashed character matches itself

- Expressions within parentheses are matched as subpattern groups and saved for use by certain methods.

By default, a quantified subpattern is *greedy* . In other words it matches as many times as possible without causing the rest of the pattern not to match. To change the quantifiers to match the minimum number of times possible, without causing the rest of the pattern not to match, you may use a "?" right after the quantifier.

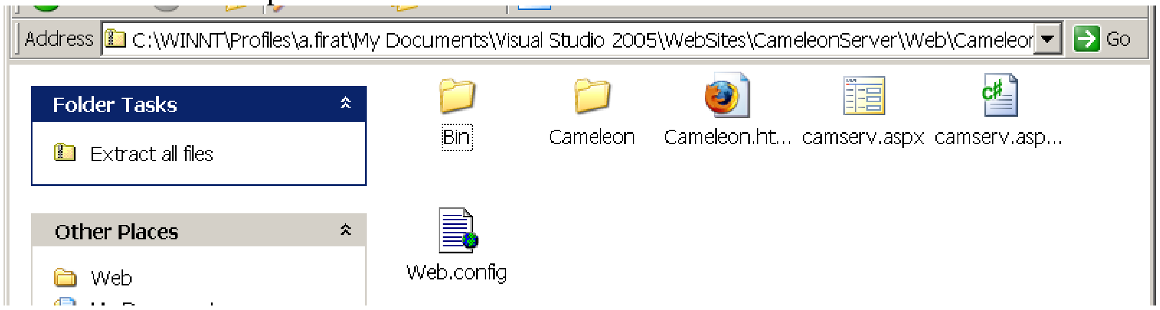
*? : Match 0 or more times
+? : Match 1 or more times
?? : Match 0 or 1 time
{n}? : Match exactly n times
{n,}? : Match at least n times
{n,m}? : Match at least n but not more than m times

Perl5 extended regular expressions are fully supported.

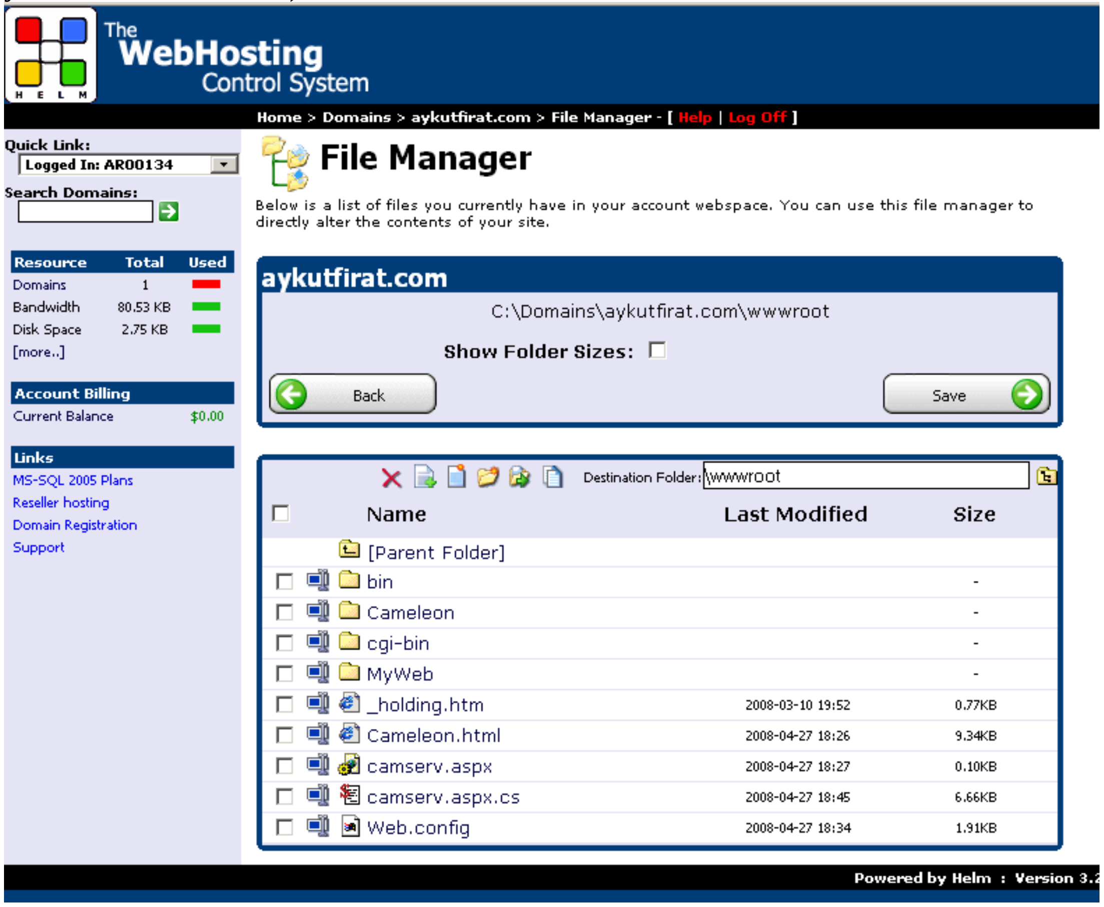
(?#text) : An embedded comment causing text to be ignored.
(?:regexp) : Groups things like "()" but doesn't cause the group match to be saved.
(?=regexp) : A zero-width positive lookahead assertion. For example, \w+(?=\s) matches a word followed by whitespace, without including whitespace in the MatchResult.
(?!regexp) : A zero-width negative lookahead assertion. For example foo(?!bar) matches any occurrence of "foo" that isn't followed by "bar". Remember that this is a zero-width assertion, which means that a(?!b)d will match ad because a is followed by a character that is not b (the d) and a d follows the zero-width assertion.
(?imsx) : One or more embedded pattern-match modifiers. i enables case insensitivity, m enables multiline treatment of the input, s enables single line treatment of the input, and x enables extended whitespace comments.

A.3 Installing Cameleon# Server

1. Get Cameleon.zip file. Its contents should look like:



2. Upload these files to a server that supports .NET framework. (In the snapshot below cgi-bin, MyWeb, and _holding.htm were preexisting directories and file, you do not need them.)



Below is a list of files you currently have in your account webspace. You can use this file manager to directly alter the contents of your site.

Resource	Total	Used
Domains	1	🟥
Bandwidth	80.53 KB	🟩
Disk Space	2.75 KB	🟩
[more..]		

Current Balance $0.00

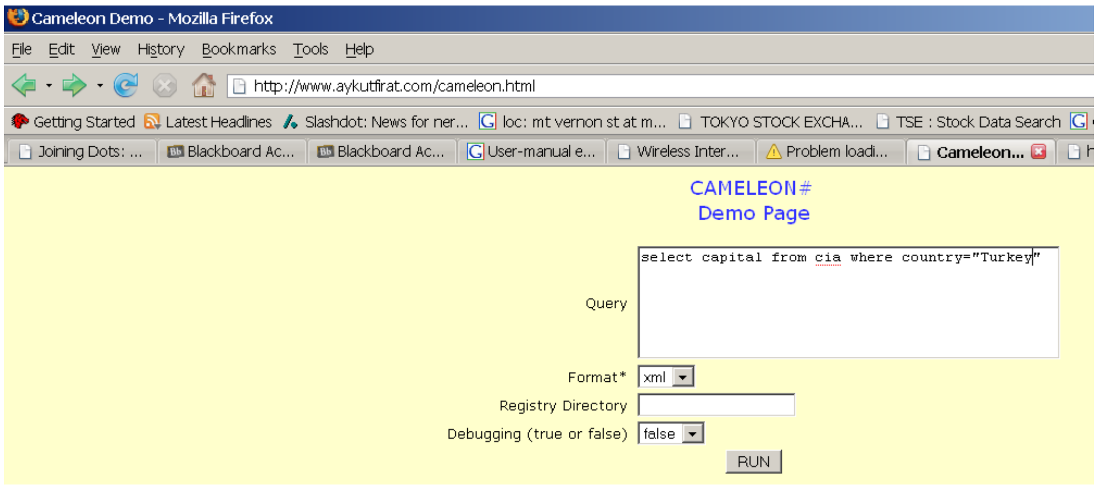
aykutfirat.com
C:\Domains\aykutfirat.com\wwwroot

Destination Folder: \wwwroot

Name	Last Modified	Size
[Parent Folder]		
bin		-
Cameleon		-
cgi-bin		-
MyWeb		-
_holding.htm	2008-03-10 19:52	0.77KB
Cameleon.html	2008-04-27 18:26	9.34KB
camserv.aspx	2008-04-27 18:27	0.10KB
camserv.aspx.cs	2008-04-27 18:45	6.66KB
Web.config	2008-04-27 18:34	1.91KB

3. Spec files go under the Cameleon directory.

4. Test by using Cameleon.html

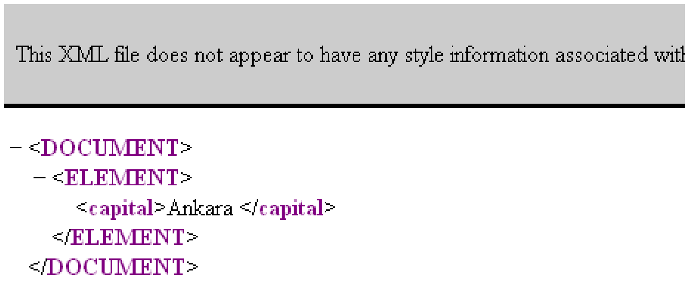

Cameleon Demo - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://www.aykutfirat.com/cameleon.html
CAMELEON#
Demo Page
RUN

Query: `select capital from cia where country="Turkey"`

Format*: xml

Registry Directory:

Debugging (true or false): false



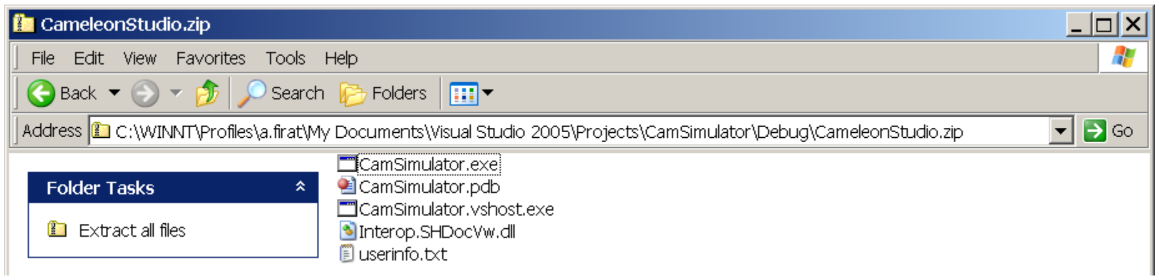
This XML file does not appear to have any style information associated wit

```
- <DOCUMENT>
  - <ELEMENT>
      <capital>Ankara </capital>
    </ELEMENT>
  </DOCUMENT>
```

5. Now you can author spec files, upload them into your spec file directory, and run.

A.4 Installing Cameleon# Studio

1. Download CameleonStudio.zip



C:\WINNT\Profiles\a.firat\My Documents\Visual Studio 2005\Projects\CamSimulator\Debug\CameleonStudio.zip

CamSimulator.exe
CamSimulator.pdb
CamSimulator.vshost.exe
Interop.SHDocVw.dll
userinfo.txt

2. Run CamSimulator.exe

   If you have any difficulties, make sure you have the latest .NET framework components.