



CUAHSI
universities allied for water research

HIS document 5

CUAHSI WaterOneFlow Workbook (version 1.1)

**A guide to using CUAHSI's WaterOneFlow web services
to retrieve hydrologic time series data**

January 2010

Prepared by:

Tim Whiteaker

**Center for Research in Water Resources
University of Texas at Austin**

Distribution

Copyright © 2008 University of Texas at Austin

CUAHSI's WaterOneFlow web services are documented at the following URL:
<http://his.cuahsi.org/wofws.html>

Disclaimers

Although much effort has been expended in the development and testing of the WaterOneFlow, errors and inadequacies may still occur. Users must make the final evaluation as to the usefulness of WaterOneFlow for his or her application.

Acknowledgements

The team of engineers, scientists and research assistants that contributed to this document includes:

Tim Whiteaker (editor), Research Associate, Center for Research in Water Resources, University of Texas, Austin, TX.

David Tarboton, Professor, Civil and Environmental Engineering, Utah State University, Logan, UT.

Jon Goodall, Assistant Professor, Nicholas School of the Environment and Earth Sciences, Duke University, Durham, NC.

David Valentine, GIS Programmer, San Diego Supercomputer Center, University of California at San Diego, La Jolla, CA.

Ernest To, Doctoral Candidate, Center for Research in Water Resources, University of Texas, Austin, TX.

Bora Beran, Doctoral Candidate, Computational Hydraulics Lab, Drexel University, Philadelphia, PA.

Funding

Funding for this document was provided by the Consortium of Universities for the Advancement of Hydrologic Science, Inc. (CUAHSI) under NSF Grant No. EAR-0413265. In addition, much input and feedback has been received from the CUAHSI Hydrologic Information System development team. Their contribution is acknowledged here.

Table of Contents

Distribution	ii
Disclaimers	ii
Acknowledgements	ii
1.0 Introduction	1
1.1 WaterOneFlow Web Services	1
1.2 WaterOneFlow Web Service Methods and Output.....	2
1.2.1 GetSiteInfo/GetSiteInfoObject	3
1.2.2 GetVariableInfo/GetVariableInfoObject	3
1.2.3 GetValues/GetValuesObject	4
1.3 Document Outline	5
1.4 Obtaining This Workbook.....	6
2.0 Data Sources	7
2.1 USGS National Water Information System (NWIS)	7
2.2 Moderate Resolution Imaging Spectroradiometer (MODIS).....	7
3.0 Connecting Excel to WaterOneFlow with HydroExcel.....	9
4.0 Ingesting Weather and Streamflow Data into ArcGIS with HydroGET.....	12
5.0 Plotting MODIS Data with Matlab	14
5.1 Introduction	14
5.2 Computer and Skill Requirements	14
5.3 Procedure.....	14
5.3.1 Setting up the XML Parser	14
5.3.2 Retrieving MODIS Data	15
6.0 Ingesting NWIS Data using VB.Net	21
6.1 Introduction	21
6.2 Computer and Skill Requirements	21
6.3 Accessing NWIS Data with a VB.Net Windows Application	21
6.3.1 Setting up the Project	21
6.3.2 Creating the Web Reference	22
6.3.3 Building the User Interface.....	23
6.3.4 Writing the Code.....	26

6.3.5	Running the Code	28
7.0	Ingesting NWIS Data Using Java	29
7.1	Computer and Skill Requirements	29
7.2	Procedure.....	29
7.2.1	Creating a New Project	29
7.2.2	Creating a Web Service Client.....	30
7.2.3	Creating a Class to Consume the Web Service.....	32
Appendix A:	Source Code for parse_xml.m	40
Appendix B:	Source Code for MODISPlot_xml.m	46
Appendix C:	Source Code for nwis.java Class.....	48

1.0 Introduction

One of the key programs of the Consortium of Universities for the Advancement of Hydrologic Science (CUAHSI) is the development of Hydrologic Information Systems (HIS), which facilitate the integration of data and software to support hydrologic science. A main component of CUAHSI HIS is WaterOneFlow web services, which provide programmatic access to a growing collection of national, state, and individual investigator hydrologic observation repositories. This document describes how to use WaterOneFlow web services and methods, with tutorials providing examples of data access in a variety of software environments.

1.1 WaterOneFlow Web Services

Wikipedia gives the following definition for a web service:

According to the W3C a Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface that is described in a machine-processable format such as WSDL. Other systems interact with the Web service in a manner prescribed by its interface using messages, which may be enclosed in a SOAP envelope, or follow a RESTful approach.

When a web service is published on the Internet, a computer with an Internet connection can call upon the web service to perform useful work.

CUAHSI WaterOneFlow web services facilitate the retrieval of hydrologic observations data. While several repositories of national hydrologic data are already available online, each data provider uses its own methodology for querying data, and its own output format for returning data. WaterOneFlow web services provide a common methodology and output format for these data sources, and permit data access directly from within the user's preferred software environment, rather than requiring the user to navigate to the data provider's web page, query data, and save the data locally.

WaterOneFlow web services have been developed for the following national networks:

USGS National Water Information System (NWIS) – national database of streamflow, water quality, and groundwater data

<http://water.sdsc.edu/waterOneFlow/NWIS/DailyValues.asmx>

<http://water.sdsc.edu/waterOneFlow/NWIS/UnitValues.asmx>

<http://water.sdsc.edu/waterOneFlow/NWIS/Data.asmx>

<http://water.sdsc.edu/waterOneFlow/NWIS/Groundwater.asmx>

EPA STORET – national database of water quality data

http://water.sdsc.edu/waterOneFlow/EPA/cuahsi_1_0.asmx

Daymet – daily surfaces of temperature, precipitation, humidity, and radiation for the contiguous United States

<http://water.sdsc.edu/waterOneFlow/DAYMET/Service.asmx>

MODIS – remotely sensed meteorological, oceanographic, and hydrologic data for the world
<http://water.sdsc.edu/waterOneFlow/MODIS/Service.asmx>

North American Mesoscale model (NAM) – prediction of climate variables for North America
<http://water.sdsc.edu/waterOneFlow/NAM12k/Service.asmx>

To access a catalog of networks currently available through WaterOneFlow, visit HIS Central at
<http://water.sdsc.edu/centralhis/>.

1.2 WaterOneFlow Web Service Methods and Output

To standardize access to these data sources, WaterOneFlow web services implement the following core methods regardless of data provider:

- GetSiteInfo
- GetSiteInfoObject
- GetVariableInfo
- GetVariableInfoObject
- GetValues
- GetValuesObject

In addition to consistent method names, WaterOneFlow services use consistent method signatures, and provide output in a consistent format, regardless of data provider. Thus, if you learn how to use the WaterOneFlow services for NWIS, you should easily be able to make the jump to using the WaterOneFlow service for EPA STORET.

For the methods with the suffix “Object” in the method name, data are returned in object format. For the other methods, data are returned in XML format. Both Object and XML formats are provided to suit the developer’s preference, or to accommodate the application programming environment of the developer.

NOTE: All methods include an authorization token parameter (authToken). This token permits CUAHSI to restrict access to web services. For the services described in this workbook, any authorization token will work, as all of these services are publicly available.

General documentation about CUAHSI web services can be found at
<http://his.cuahsi.org/wofws.html>.

Below is a brief summary of the core WaterOneFlow methods.

1.2.1 *GetSiteInfo/GetSiteInfoObject*

This method returns basic information about a site, such as its name, location, and a list of variables available at the site for time series retrieval. The method has the following signature:

```
GetSiteInfo(String location, String authToken)
```

Input Parameters:

location

For networks which measure data at discrete sites, such as NWIS, this location parameter should be written as “NetworkName:SiteCode”. For example, to specify the NWIS stream gage at the Colorado River at Austin (site code 08158000), use the following value for the location parameter:

```
"NWIS:08158000"
```

For networks that return a time series for any point in space (such as Daymet), use the convention “GEOM:POINT(Longitude Latitude)”. For example, to specify a point at 113 degrees West Longitude and 45 degrees North Latitude, use the following value for the location parameter:

```
"GEOM:POINT(-113 45) "
```

For networks that return a time series for a box defined by Latitude and Longitude coordinates (such as MODIS), use the convention “GEOM:BOX(WestLongitude SouthLatitude, EastLongitude NorthLatitude)”. For example, to specify a box that covers the whole earth, use the following value for the location parameter:

```
"GEOM:BOX(-180 -90,180 90) "
```

authToken

This parameter allows a data provider to restrict or monitor access to its web services, by requiring a password or some other means of identification in order to use a given web method. In many cases, this parameter may be left as an empty string, “”.

Example VB.Net Code

```
Dim ws As New [WsReference]  
Dim result As String = ws.GetSiteInfo("NWIS:08158000", "")  
Debug.Write(result)
```

1.2.2 *GetVariableInfo/GetVariableInfoObject*

This method returns information about a time series variable, such as name and units. The method has the following signature:

```
GetVariableInfo(String variable, String authToken)
```

Input Parameters:

variable

To specify a variable, you must include both the network name and the variable code within the network, in the following format: “NetworkName:VariableCode”. For example, to query the NWIS website for information about variable code 00010 (which happens to be water temperature), you would use the following value for the variable parameter:

```
"NWIS:00010"
```

authToken

This parameter allows a data provider to restrict or monitor access to its web services, by requiring a password or some other means of identification in order to use a given web method. In many cases, this parameter may be left as an empty string, “”.

Example VB.Net Code

```
Dim ws As New [WsReference]  
Dim result As String = ws.GetVariableInfo("NWIS:00010", "")  
Debug.Write(result)
```

1.2.3 GetValues/GetValuesObject

This method returns a time series for a given variable at a given location. The method has the following signature:

```
GetValues(String location, String variable, String startDate, String endDate,  
String authToken)
```

Input Parameters:

location

The location parameter is the same as for the GetSiteInfo method.

variable

The variable parameter is the same as for the GetVariableInfo method, except that this parameter may also include options for data retrieval. The parameter should be specified as follows:

```
"NetworkName:VariableCode/Option=Value"
```

In most cases, no option is required, and the “NetworkName:VariableCode” format may be used. Some networks, such as MODIS, do require an option to be set. As an example, to retrieve data for Cloud Optical Thickness in the Water Phase from MODIS, for a spatial average that includes both the land surface and the ocean, you would use the following value for the **variable** parameter:

```
"MODIS:11/plotarea=landocean"
```

startDate

This parameter specifies the start datetime for which time series records are desired. For networks that return time series with a temporal precision of one day or longer, use the following format for the startDate:

"yyyy-mm-dd"

For example, to specify the last day of 2003 as the start date for time series retrieval, use the following value for the startDate parameter:

"2003-12-31"

For networks with a temporal precision shorter than one day, you may specify the hours and minutes and so on with the format below:

"yyyy-mm-ddThh:mm:ss"

For example, to specify 6:30 AM on the last day of 2003, use the following value:

"2003-12-31T06:30"

endDate

This parameter specifies the end datetime for which time series records are desired. The format is the same as for the startDate parameter.

authToken

This parameter allows a data provider to restrict or monitor access to its web services, by requiring a password or some other means of identification in order to use a given web method. In many cases, this paramter may be left as an empty string, "".

Example VB.Net Code

```
Dim ws As New [WsReference]
Dim result As String = ws.GetValues("NWIS:08158000", _
    "NWIS:00010", _
    "2003-01-01", _
    "2003-12-31", _
    "")
Debug.Write(result)
```

1.3 Document Outline

The document is created in the form of a series of tutorials. The tutorials show how to use various software or programming environments to access different WaterOneFlow web services and methods. The links for obtaining installation and data files for each tutorial are provided in the tutorial. In the current version of the document, you will learn how to access data from USGS NWIS and MODIS Matlab, VB.Net and Java. Web service access from Excel and ArcGIS is also described, with links to specific software products built to facilitate those connections. The outline of this document is as follows:

Chapter 1 – Introduction

Chapter 2 – Data Sources

Chapter 3 – Ingesting data into Excel (Introduction to HydroObjects and HydroExcel)

Chapter 4 – Ingesting data into ArcGIS (Introduction to HydroGET)

Chapter 5 – Ingesting data into Matlab (MODIS example)
Chapter 6 – Ingesting data using VB.Net (NWIS Unit Values example)
Chapter 7 – Ingesting data using Java (NWIS Daily Values example)

1.4 Obtaining This Workbook

This workbook is available at the following location:

http://his.cuahsi.org/documents/HISDoc5_UseWebServices.pdf

2.0 Data Sources

This chapter describes the data sources behind WaterOneFlow web services used in this workbook.

2.1 USGS National Water Information System (NWIS)

Data providing organization: United States Geological Survey (USGS)

Website: <http://waterdata.usgs.gov/nwis>

The USGS NWIS is a comprehensive and distributed program that supports acquisition, processing and storage of water data. Most of the data stored in NWIS is available through NWIS website provided above (NWIS Web). The data available via NWIS web mainly include information on quantity and quality of surface and ground water. NWIS web serves both historical and real time data. The real time data, however, is not available for all sites.

Data provided by NWISWeb are regularly updated from NWIS. Real-time data are generally updated upon receipt at local Water Science Centers. NWISWeb provides access to data by category, such as surface water, ground water, or water quality, and by geographic area. NWIS data are available for all 50 states, plus border and territorial sites, and include data from as early as 1899 (at few stations) to present. Of the over 1.5 million sites with NWIS data, the vast majority (about 800,000) are for groundwater wells, about 25,000 sites are for streamflow data, and about 9,800 of the sites provide real-time data. In addition there are many sites with atmospheric data such as precipitation, and there are nearly 70 million water-quality results from about 4 million water samples collected at hundreds of thousands of sites.

2.2 Moderate Resolution Imaging Spectroradiometer (MODIS)

Data providing organization: National Aeronautics and Space Administration (NASA)

Website: http://g0dup05u.ecs.nasa.gov/Giovanni/modis.MOD08_M3.shtml

The Goddard Earth Sciences Data and Information Services Center (GES DISC) has created the GES DISC Interactive Online Visualization and Analysis Infrastructure (Giovanni) to enable Web-based visualization and analysis of satellite remotely sensed meteorological, oceanographic, and hydrologic data. The MODIS data are available through one of the Giovanni interfaces called the MODIS Online Visualization and Analysis (MOVAS). The MOVAS system, operational since September 2003, provides access to download, visualize and analyze MODIS Level-3 atmospheric monthly products. The MODIS Level-3 data includes monthly 1 x 1 degree grid average values of atmospheric parameters related to atmospheric aerosol particle properties, total ozone burden, atmospheric water vapor, cloud optical and physical properties, and atmospheric stability indices.

The data are available for the entire globe from March 1, 2000, to typically six months to a year prior to the current date. The time series returned by MOVAS is spatially averaged over the

extent specified by a bounding box of lat-long coordinates. The data are temporally averaged with a monthly time step.

3.0 Connecting Excel to WaterOneFlow with HydroExcel

While WaterOneFlow web services support standardized, automated queries for hydrologic data, they may be difficult to use for those who are uninitiated into the world of web services. Therefore, CUAHSI HIS includes extensions to applications commonly used in hydrologic science in order to connect those applications with WaterOneFlow. One of these applications is HydroExcel, which utilizes an object library called HydroObjects in order to make dynamic connections to web services.

HydroObjects

<http://his.cuahsi.org/hydroobjects.html>

HydroObjects is a .NET DLL with COM classes that support hydrology applications. The key class in the library is WebServiceWrapper, which provides a method for calling Web Services from a COM (e.g., Visual Basic for Applications (VBA)) environment. This class can be used to call WaterOneFlow web services for downloading hydrologic time series.

HydroExcel - WaterOneFlow in Excel

<http://his.cuahsi.org/hydroexcel.html>

HydroExcel is a Microsoft Excel spreadsheet that uses macros and HydroObjects to download hydrologic observations data from WaterOneFlow web services. This means that you can query for observation sites, variables, and time series data from online resources directly within Excel. As long as a web service follows WaterOneFlow specifications, HydroExcel will be able to communicate with it. Thus, HydroExcel provides a window into the nation's water data from within one of the most widely used applications within the hydrologic science community. HydroExcel consists of nine worksheets, eight of which provide access to WaterOneFlow web services. Each worksheet accesses a specific kind of data. For example, the figure below shows a screenshot of the Time Series worksheet, which is used for downloading a time series of values for a given variable at a given location.

GetValues		<input checked="" type="checkbox"/> Ignore NoData Value
Site Code/Location	NWIS:08158000	
Variable Code	NWIS:00060	
Start Date	5/1/2008 0:00	
End Date	6/30/2008 0:00	
<div>Get Values</div>		

To use HydroExcel, you indicate the web service that you want to work with, and then click buttons in the spreadsheet to download information from the web service. Links to some

existing WaterOneFlow web services are provided in the spreadsheet to get you started, as well as informative text. A screenshot of the Data Source worksheet is shown below.

Data Source

In the box next to **WSDL Location**, input (hint: copy and paste) the WSDL location for the WaterOneFlow web service you want to access.

This web service will be used in all other worksheets.

WSDL Location:

Click **Get Capabilities** to see what functionality is available with the web service.

Some services may not support all the methods that this spreadsheet utilizes. For example, consider a service that provides access to a gridded dataset. Since the data are based on a grid, and not on discrete sensor locations or "sites", the service will not support a GetSites or GetSiteInfo method, and so you will not be able to use the Sites, Site Info, or Site Catalog worksheets with that service.

Click **Open Service Web Page** to open a web page that may have more information about the web service.

Click **Get Sites** to jump to the Sites worksheet and download a list of sites from the web service.

Click **Get Variables** to jump to the Variables worksheet.

Web Services for National Data Sources

Data Source	WSDL Location	Description
United States Geological Survey	http://river.sdsc.edu/wateroneflow/NWIS/DailyValues.asmx?WSDL	NWIS daily values
United States Geological Survey	http://river.sdsc.edu/wateroneflow/NWIS/Groundwater.asmx?WSDL	NWIS groundwater
United States Geological Survey	http://river.sdsc.edu/wateroneflow/NWIS/UnitValues.asmx?WSDL	NWIS real time
United States Geological Survey	http://river.sdsc.edu/wateroneflow/NWIS/Data.asmx?WSDL	NWIS instantaneous
Oak Ridge National Laboratory	http://river.sdsc.edu/wateroneflow/DAYMET/Service.asmx?WSDL	Daymet Meteorological
National Centers for Environmental Prediction	http://river.sdsc.edu/wateroneflow/NAM12k/Service.asmx?WSDL	North American Monsoon
Environmental Protection Agency	http://river.sdsc.edu/wateroneflow/EPA/cuahsi_1_0.asmx?WSDL	STORET water quality
NASA	http://river.sdsc.edu/wateroneflow/MODIS/Service.asmx?WSDL	Atmospheric MODIS

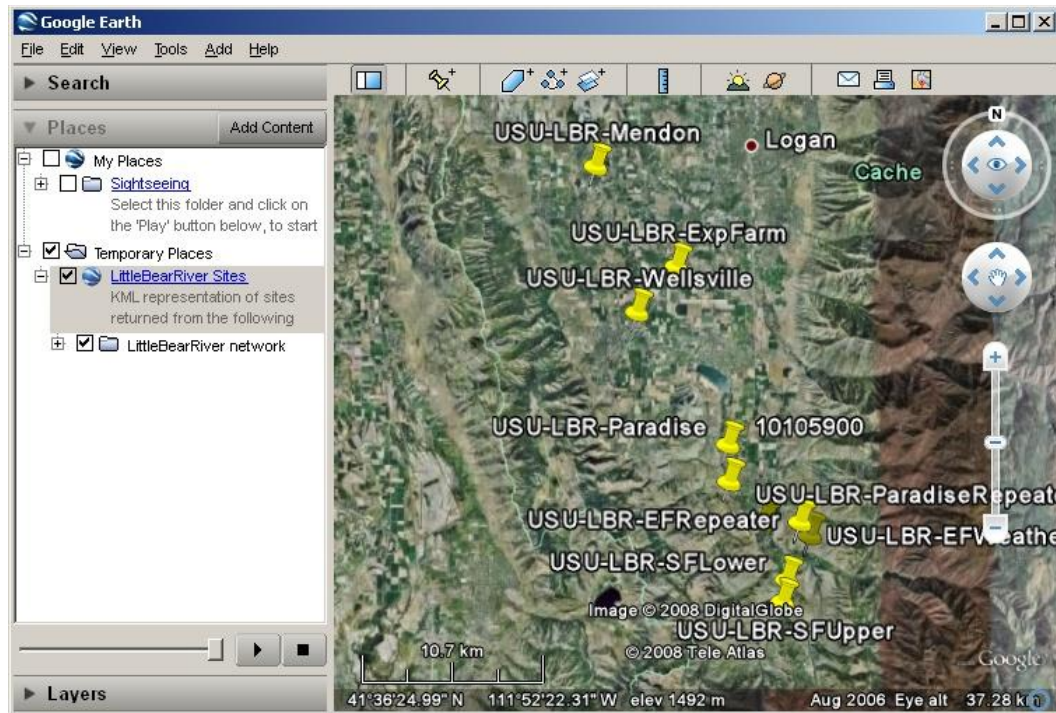
Web Services for Academic Investigator Data

University	WSDL Location	Description
Utah State University	http://his02.usu.edu/littlebearriver/cuahsi_1_0.asmx?WSDL	Utah State University
Utah State University	http://his02.usu.edu/mudlake/cuahsi_1_0.asmx?WSDL	Utah State University
University of Iowa	http://his06.ihr.uiowa.edu/nexrad/cuahsi_1_0.asmx?WSDL	University of Iowa
University of Iowa	http://his06.ihr.uiowa.edu/water_quality/cuahsi_1_0.asmx?WSDL	University of Iowa
University of Iowa	http://his06.ihr.uiowa.edu/lincoln/cuahsi_1_0.asmx?WSDL	University of Iowa

The worksheets and their functions are:

- **Introduction** – Introduce the worksheet and provide license information.
- **Data Source** – Set the web service that will be accessed in the spreadsheet.
- **Sites** – Download a list of sites available from the web service.
- **Variables** – Download a list of variables available from the web service.
- **Site Info** – Download information about a specific site, including a list of variables measured at the site.
- **Site Catalog** – Download site info for several sites at once.
- **Site Summary** – Use a pivot table to summarize the site catalog, typically by the number of values of a given variable measured at each site.
- **Time Series** – Download a time series of values for a given variable at a given location for a given time period.
- **Statistics and Charts** – Use a pivot table and chart to summarize time series data.

When downloading site information, HydroExcel can build a KML file to show the sites in Google Earth. This provides a spatial component that complements the tabular nature of the Excel spreadsheet. A screenshot of observations sites for the Little Bear River network is shown in the figure below.



In summary, HydroExcel provides access to WaterOneFlow web services from the Excel application environment using HydroObjects and VBA macros. To download HydroExcel installation files and the software manual (with tutorial), visit <http://his.cuahsi.org/hydroexcel.html>.

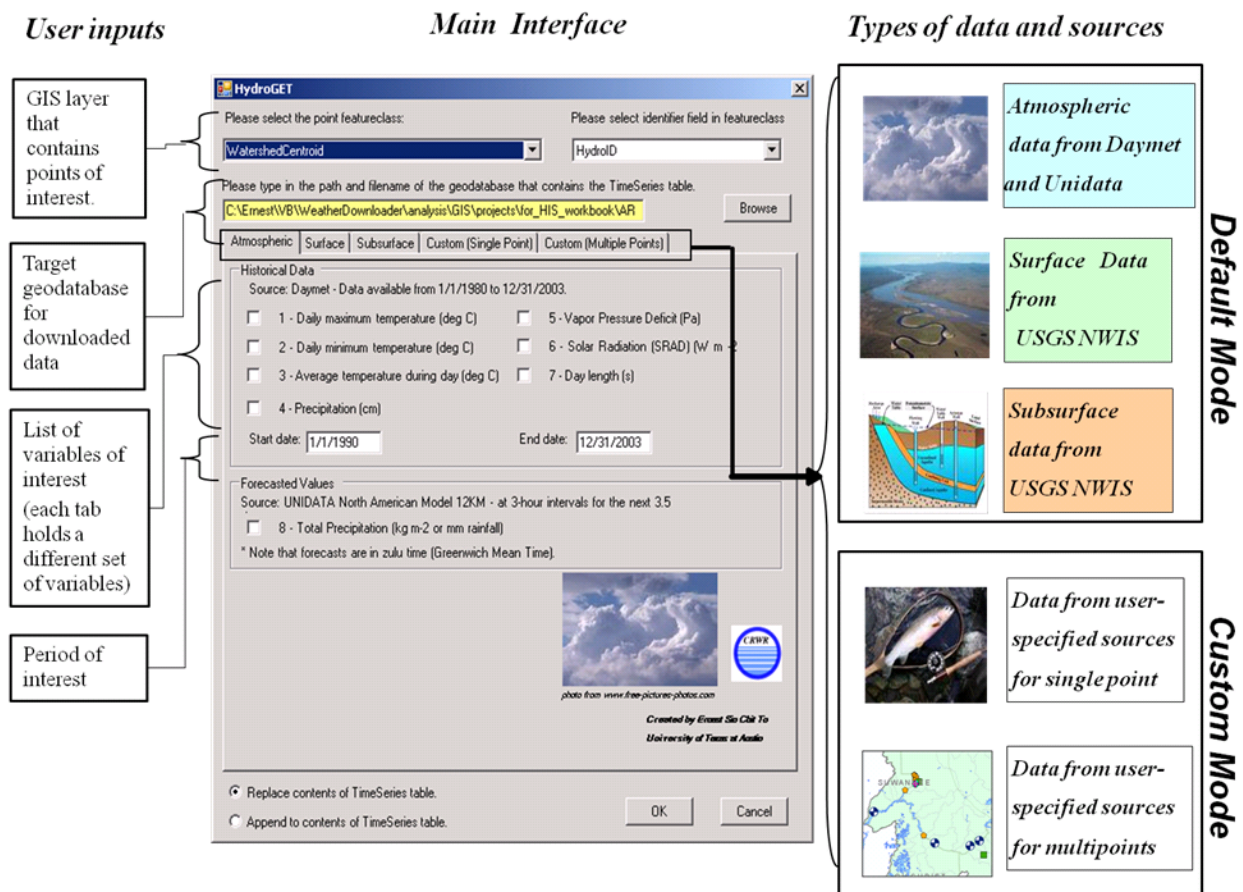
4.0 Ingesting Weather and Streamflow Data into ArcGIS with HydroGET

<http://his.cuahsi.org/hydroget.html>

CUAHSI's HydroGET (**H**ydrologic **G**IS **E**xtraction **T**ool) is a versatile tool that gives ArcGIS users the ability to ingest web service data into ArcGIS. HydroGET stores the downloaded data using the time series format used in the well-known data model, Arc Hydro, and its upcoming successor, Arc Hydro II. HydroGET can work with any web service as long as the web service complies with CUAHSI's WaterML protocol.

The figure below shows the program interface of HydroGET. It contains five tabs for the user to specify what kind of environmental data he wants to download. These tabs are listed as follows:

- Atmospheric
- Surface
- Subsurface
- Custom (single point); and,
- Custom (multiple points).



The **Atmospheric, Surface, Subsurface** tabs offer the user the ability to download data from preset web services to describe the different component of the hydrologic cycle. These web services access Daymet and Unidata for atmospheric data and USGS NWIS for surface water and groundwater data. Together they enable the user to characterize the hydrologic characteristics of his geographical area of interest.

When the user utilizes only these three tabs to download data, HydroGET is operating in ***Default Mode***. This means that HydroGET only calls the web-services and variables that have already been hard-wired into its code. When HydroGET is operating in ***Default Mode***, it is very user-friendly and does not require the user to have any background knowledge of web services. However, its capabilities in this mode are limited as it cannot handle data sources other than Daymet, Unidata and NWIS.

The **Custom (single point)** and **Custom (multiple points)** tabs allow the user to retrieve data from any user-defined web services that comply with WaterML format. When the user utilizes these two tabs to download data, HydroGET is operating in ***Custom Mode***. ***Custom Mode*** requires the user to have slightly more knowledge about web services and is recommended for the intermediate user. In this mode HydroGET becomes truly a powerful tool. Not only does it have the ability to access a wide range of web services, it also has the ability to batch process multiple requests to different web services. In this mode it essentially becomes a harvester for data.

For the HydroGET installation files and tutorial, see <http://his.cuahsi.org/hydroget.html>.

5.0 Plotting MODIS Data with Matlab

by David Tarboton

5.1 Introduction

Matlab users can take advantage of web service methods by using the *createClassFromWSDL* function. This function creates a Matlab class based on a WSDL. The URL to the WSDL is provided to the function when the function is called. This chapter demonstrates how to call a CUASHI web service from Matlab, parse the result, and plot a time series graph. In the exercise, you will write an M-file that creates a plot of the Cloud Optical Thickness Water Phase variable from NASA's MODIS database of remote sensing data.

5.2 Computer and Skill Requirements

To complete this exercise, your computer must meet the following requirements:

- Working Internet connection
- Matlab version 7 (or greater) software

This exercise assumes that you have some familiarity with the following software environments:

- Matlab version 7

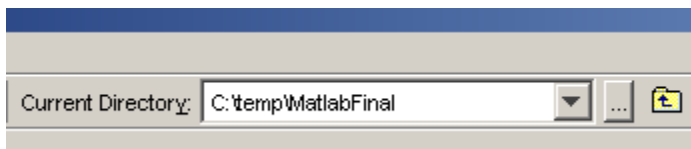
NOTE: The source code for the *parse_xml* M-file used in this exercise is located in Appendix A. The source code for the *MODISPlot_xml* M-file is located in Appendix B.

5.3 Procedure

WaterOneFlow web services return data either in XML or Object form. XML is a very useful format for web services, because it is platform independent and self-describing. Yet while Matlab can create a class from a WSDL, it does not contain inherent classes for working with XML. Therefore, we'll begin the exercise by creating an M-file called *parse_xml* that will serve as our Matlab XML parser.

5.3.1 Setting up the XML Parser

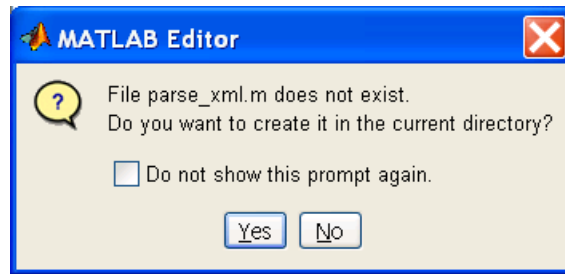
1. Start Matlab.
2. Set the current directory to the location where you wish to perform the work.



3. If you already have a copy of the *parse_xml.m* file, copy the file to the working directory and go to the section entitled Retrieving MODIS Data. Otherwise, continue to the next step.
4. In the Command Window, enter the command

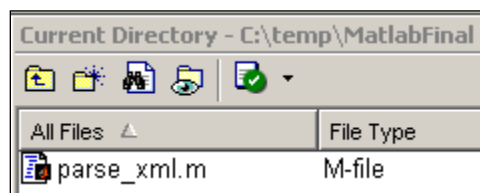
```
edit parse_xml
```

If you get the following prompt, click Yes to create the file parse_xml.



5. In the Matlab editor, enter the code for parse_xml as found in Appendix A. If you are viewing an electronic copy of this document, try copying and pasting the code.
6. In the Matlab editor, click the File menu, and then click Save.

You should now see the parse_xml.m file in Matlab's Current Directory window.



5.3.2 Retrieving MODIS Data

With the XML parser in place, the next step is to build an M-file for retrieving MODIS data.

NOTE: Instead of entering code as shown below, you could copy the code from Appendix B, or simply refer to MODISPlot_xml.m if you were provided a copy of the finished file with this document.

1. In the Command Window, enter the command

```
edit MODISPlot_xml
```

2. In the Matlab editor, enter the following lines of code. This code creates an instance of the MODIS class from the WSDL.

```
% Create class.  
wsdl='http://water.sdsc.edu/waterOneFlow/MODIS/Service.asmx?WSDL';  
createClassFromWsd1(wsdl);  
  
% This creates an instance of the class.  
svsMODIS = MODIS;
```

3. In the Matlab editor, enter the following lines of code. This code sets the parameters for calling the GetValues method, and then calls GetValues. In this case, the parameters instruct the MODIS class to retrieve Cloud Optical Thickness Water Phase values for 2004, spatially averaged over an area that roughly covers Travis County in Texas. The plotArea parameter indicates that the values should include areas both over the land surface and the oceans. You will find a list of valid codes and keywords for retrieving MODIS data at:

<http://water.sdsc.edu/waterOneFlow/MODIS/Service.asmx>

```
% Specify input parameters.
w='-98.2' % West longitude.
s='30' % South latitude.
e='-97.3' % East longitude.
n='30.7' % North latitude.
location=['GEOM:BOX(',w,',',s,',',e,',',n,')']
% Variable Code 11 = Cloud Optical Thickness Water Phase.
variableCode='MODIS:11/plotarea=land'
startDate='2004-01-01'
endDate='2004-12-01'

% Call the GetValues function to get the time series data.
xmlValues=GetValues(svsMODIS,location,variableCode, ...
    startDate,endDate, '')
```

4. In the Matlab editor, click the File menu, and then click Save.
5. In the Matlab Command Window, enter the command

MODISPlot_xml

This command runs the code in the MODISPlot_xml.m file. When the code runs, you will see the values that have been set for the parameters to the GetValues call, followed by the XML String returned from the web service that is saved in the variable called xmlValues.

```
endDate =

2004-12-01

xmlValues =

<timeSeriesResponse xmlns:gml="http://www.opengis.net/gml"
```

Alternatively you can run each of the commands above individually in sequence by highlighting them and pressing F9 in the Matlab editing environment.

To get an understanding of the structure of the XML output we will copy it to a file and view it using an XML viewer such as Internet Explorer.

6. Copy the XML output (beginning with `<timeSeriesResponse` and ending with `</timeSeriesResponse>`) and paste the text into a new text document using a text editor.
7. Save the document as `MODISExample.xml`, and close the text editor.
8. Open `MODISExample.xml` with an XML viewer.

Below is a screenshot of the document, as viewed in Internet Explorer. Some of the XML elements have been collapsed for readability.

```
<timeSeriesResponse xmlns:gml="http://www.opengis.net/gml" xmlns:xlink="http://www
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:wtr="http://ww
+ <queryInfo>
- <timeSeries name="MODIS">
+ <sourceInfo xsi:type="DataSetInfoType">
- <variable>
  <variableCode vocabulary="MODIS">11</variableCode>
  <variableName>Cloud Optical Thickness Water Phase (QA WT)</variableName>
  <units unitsAbbreviation="um" unitsCode="55" unitsType="Length">micron</units>
</variable>
- <values unitsAbbreviation="um" unitsCode="55" count="12">
  <value dateTime="2004-01-01T00:00:00">18.9011</value>
  <value dateTime="2004-02-01T00:00:00">31.9133</value>
  <value dateTime="2004-03-01T00:00:00">21.5478</value>
  <value dateTime="2004-04-01T00:00:00">18.8422</value>
  <value dateTime="2004-05-01T00:00:00">14.2878</value>
  <value dateTime="2004-06-01T00:00:00">9.4344</value>
  <value dateTime="2004-07-01T00:00:00">5.3655</value>
  <value dateTime="2004-08-01T00:00:00">7.3822</value>
  <value dateTime="2004-09-01T00:00:00">7.9600</value>
  <value dateTime="2004-10-01T00:00:00">10.7789</value>
  <value dateTime="2004-11-01T00:00:00">25.8700</value>
  <value dateTime="2004-12-01T00:00:00">19.5344</value>
</values>
</timeSeries>
</timeSeriesResponse>
```

Notice the hierarchy of data in the XML. Understanding the hierarchy is crucial to navigating the XML in Matlab. The `parse_xml` utility converts an XML string into a Matlab structure, the contents of which can be accessed through parent/child relationships. For example, in `MODISExample.xml`, the name of the time series variable is stored in the `variableName` tag. This tag is a child of the `variable` tag, which is a child of the `timeSeries` tag, which is a child of the `timeSeriesResponse` tag, which is a child of the XML document itself. In other words, the `variableName` tag is four levels down. The order of child tags in the same “generation” is also important. The `variable` tag is the second child of the `timeSeries` tag. Therefore, the complete path to the `variableName` tag can be summarized as follows:

- a) Get the first child (there is always only one) of the XML document. (This returns `timeSeriesResponse`)
- b) Get the second child of this element. (This returns `timeSeries`)
- c) Get the second child of this element. (This returns `variable`)
- d) Get the second child of this element. (This returns `variableName`)

This logic will be used to retrieve information from the Matlab structure created from this XML string.

9. Close the XML file.

10. In the Matlab editor for `MODISPlot_xml`, enter the following lines of code. This code calls the `parse_xml` function which feeds the XML string to the parser, and returns a Matlab structure object created from the XML string.

```
% Parse the XML string.
structValues=parse_xml(xmlValues);
```

Execute just this code by highlighting it and pressing F9. Examine the structure `structValues` that is returned, to see that it contains the complete content of the XML string in a series of nested structures. For example, if you type:

```
structValues
```

in the Matlab command window, you will see the following.

```
>> structValues

structValues =

    child: [1x1 struct]
```

This indicates that the structure returned contains one child that is itself a structure named `child`. To drill down further into this structure, the logic of the XML needs to be followed. For example

```
structValues.child.child(2).child(2).child(2)
```

returns the following

```
>> structValues.child.child(2).child(2).child(2)

ans =

    tag: 'VARIABLENAME'
  attribs: [1x1 struct]
    value: 'Cloud Optical Thickness Water Phase (QA WT)'
    child: []
```

This displays the `VARIABLENAME` tag which is the second child of the element `variable`, which is the second child of the element `timeSeries` which is the second child of the element `timeSeriesResponse` which is the only child element in the structure.

11. In the Matlab editor for `MODISPlot_xml`, enter then execute following lines of code. The display functions write the name and units for the variable to the Matlab Command Window.

```
% Report the name and units of the chosen variable.
display(structValues.child.child(2).child(2).child(2).value)
display(structValues.child.child(2).child(2).child(3).value)
```

12. In the Matlab editor for `MODISPlot_xml`, enter the following lines of code. This code retrieves the time series records from the structure then loops through all the records and stores the datetimes and values in an array. The `datenum` function converts the datetimes to numeric format, which aids in plotting the data.

```
% Get the <value> tags.
Recs=structValues.child.child(2).child(3).child;
[d1,d2]=size(Recs)

% Build arrays of datetimes and values.
for i=1:d2
    % Reformat date to that Matlab can understand it.
    datetime=Recs(i).attrs(1).value;
    year=datetime(1:4);
    month=datetime(6:7);
    day=datetime(9:10);
    datetime=[month, '/', day, '/', year];
    dn(i)=datenum(datetime); % Convert to numeric date.
    % Read the time series value.
    values(i)=str2double(Recs(i).value);
end
```

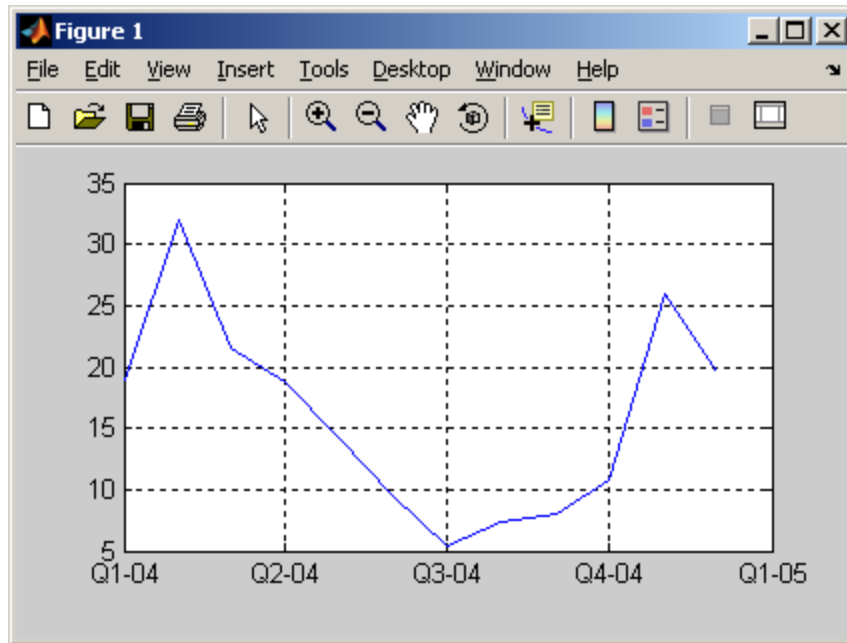
13. In the Matlab editor for `MODISPlot_xml`, enter the following lines of code. This code sets up the axis for plotting, and then plots the data using the Matlab plot function.

```
% Plot the graph.
plot(dn,values); datetick;
grid on % Turn on grid lines for this plot.
```

14. In the Matlab editor, click the File menu, and then click Save.
15. In the Matlab Command Window, enter the command

```
MODISPlot_xml
```

After a moment, you'll see the variable information appear in the Command Window, and then a graph will appear.



You can edit the plot, put legend and axes names by going to the Edit menu in Figure 1 plot shown above.

In this exercise, you have learned how to call web services from within Matlab and plot MODIS data. This concludes the exercise.

6.0 Ingesting NWIS Data using VB.Net

by Tim Whiteaker

6.1 Introduction

Using web services is a breeze with Visual Studio. This chapter demonstrates how to call a web service with Visual Studio 2008 and the Visual Basic .Net programming language.

In this exercise, you will create a VB.Net Windows application that uses the NWIS Unit Values web service to compute average streamflow over the past few days at the Colorado River at Austin, TX. The NWIS Unit Values web service returns real-time data for roughly the past 31 days. These data typically are recorded at 15-minute intervals.

6.2 Computer and Skill Requirements

To complete this exercise, your computer must meet the following requirements:

- Working Internet connection
- Visual Studio 2008 software

This exercise assumes that you have some familiarity with the following software environments:

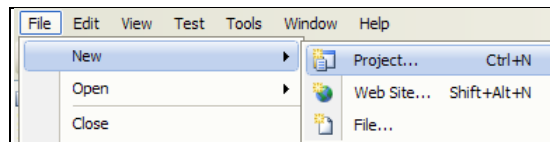
- Visual Studio 2008

6.3 Accessing NWIS Data with a VB.Net Windows Application

In this exercise, you will create a windows application with one main window that allows the user to click to see what the average streamflow over the past few days is at the Colorado River at Austin, TX. The application lets the user specify the number of days for which data should be retrieved (up to 10 days back). The application then asks the NWIS Unit Values web service for streamflow values, and then computes the average of the returned values.

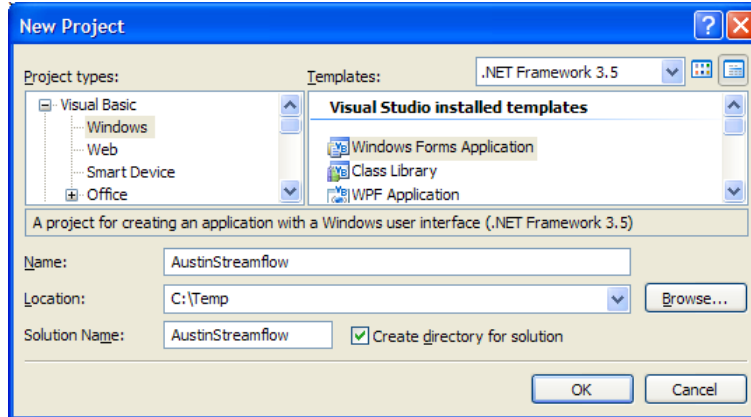
6.3.1 Setting up the Project

1. Start Visual Studio 2008 (Click on Start --- All Programs --- Microsoft Visual Studio 2008 --- Microsoft Visual Studio 2008).
2. Click the File | New | Project



3. In the New Project window, set the following properties:
 - a. Choose Visual Basic --- Windows from Project Types.
 - b. Select Windows Forms Application from Templates.
 - c. Type "AustinStreamflow" as the Name.
 - d. Set the location where you want to save the project, e.g., C:\Temp.

- e. Click OK.

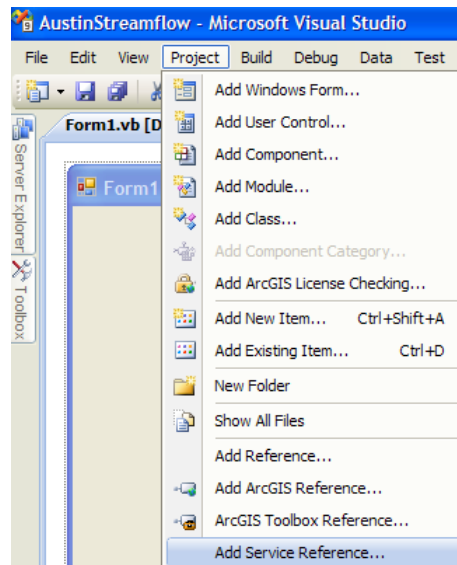


A new project will open with a default form called Form1.

6.3.2 Creating the Web Reference

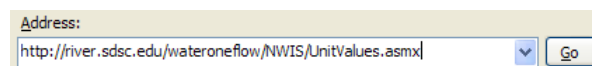
This project will make use of the NWIS Unit Values web service to retrieve streamflow values from the USGS stream gage on the Colorado River at Austin. The web service becomes available to the project after making a web reference to the service.

1. Click the Project menu, then click Add Service Reference...

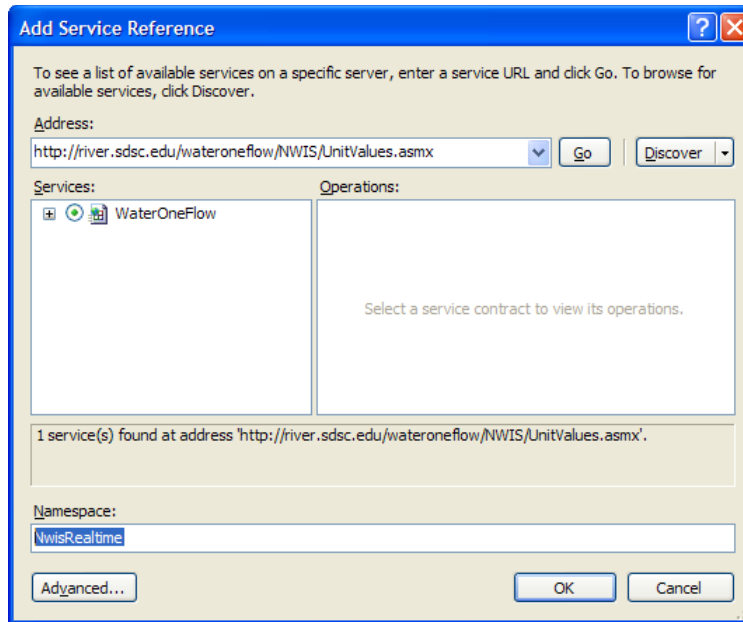


2. In the Add Service Reference window (below Address:), type in the following URL:

<http://river.sdsc.edu/wateroneflow/NWIS/UnitValues.asmx>



3. Click Go. Visual Studio will navigate to the URL and verify that a web service is present.
4. Change the namespace from the default to NwisRealtime. This is the name by which you will reference the NWIS web service in your code.



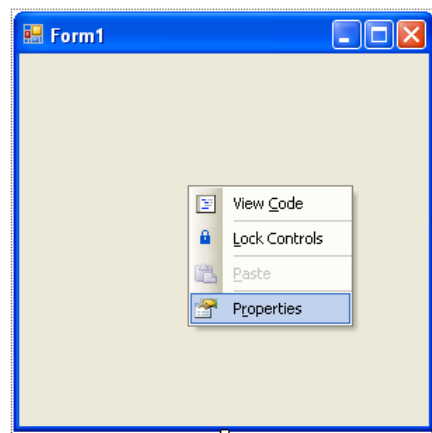
5. Click OK.

The NWIS web service is now available for use within your project.

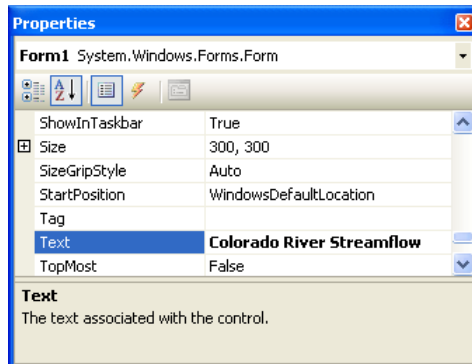
6.3.3 Building the User Interface

Now that you've set up the project, you'll build the user interface by adding controls to the form. Later, you'll add the code behind those controls which will perform the work.

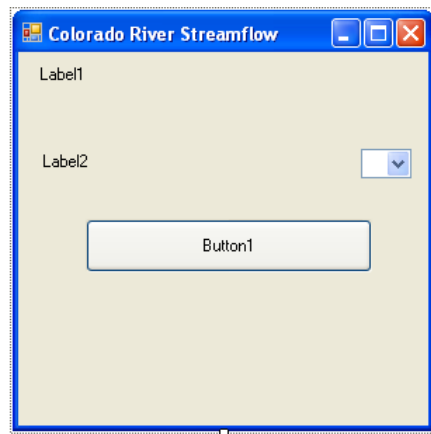
1. Right click on Form1 and click Properties.



- Change the Text property of the form to “Colorado River Streamflow”. This changes the name that appears in the title bar of the form.



- Add two labels, one combo box, and one button to the form, at roughly the same positions as shown in the figure below.

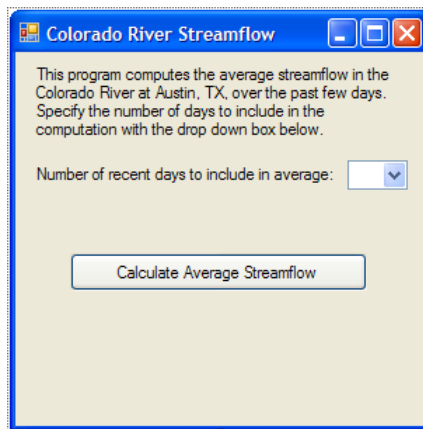


- In a manner similar to setting the Text property of the form, set the properties of the controls as shown below.

Control	Property	Value
Label1	Text	This program computes the average streamflow in the Colorado River at Austin, TX, over the past few days. Specify the number of days to include in the computation with the drop down box below.
	AutoSize	False
Label2	Text	Number of recent days to include in average:
ComboBox1	DropDownStyle	DropDownList
Button1	(Name)	btnCalculate
	Text	Calculate Average Streamflow

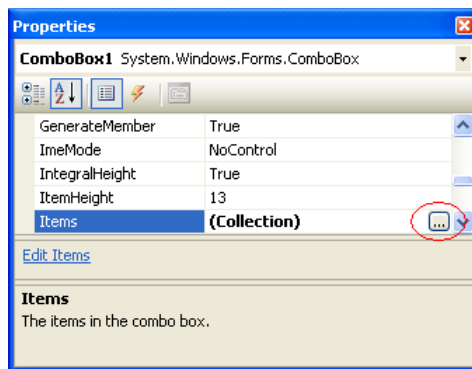
- Resize and reposition controls to make for a neat arrangement, if necessary.

The form should now look similar to the one below.

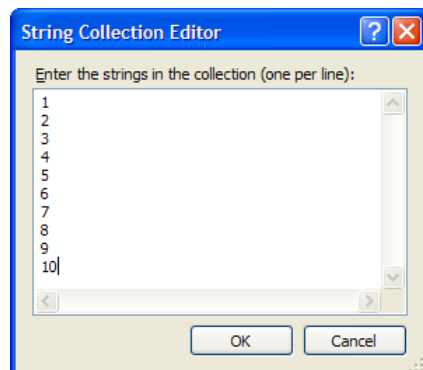


Now you will add the choice of 1 to 10 days to the combo box.

6. Click the properties for ComboBox1, and then select the Items property. Click the ellipsis next to (Collection).



7. Add the numbers 1 through 10 to the String Collection Editor window. This allows the user to select between 1 and 10 days to include in the computation of average streamflow.



8. Click OK to close the String Collection Editor window.

The design of the form is now complete. Next you will add code to make the form perform useful work.

6.3.4 Writing the Code

First you must set the default value of the drop down box. Let's use 10 as the default.

1. Double click the form (be sure and not to click on any of the controls that you have added to the form.) This opens the code editor, and creates stub code that will be run when the form opens.

```
Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object,
    End Sub
End Class
```

2. Add the following code to the Form1_Load procedure.

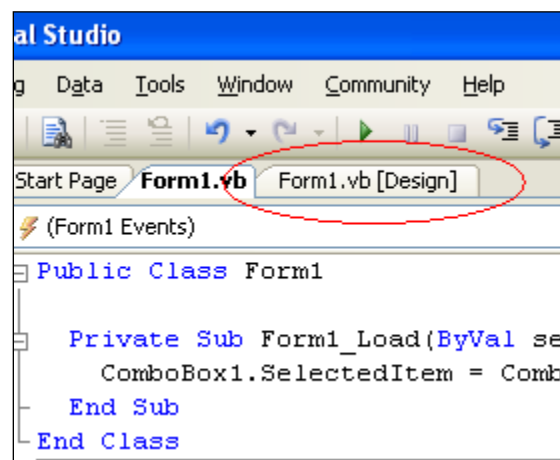
```
ComboBox1.SelectedItem = ComboBox1.Items.Item(9)
```

The result is shown in the screenshot below.

```
Private Sub Form1_Load(ByVal sender As System.Object,
    ComboBox1.SelectedItem = ComboBox1.Items.Item(9)
End Sub
```

In the code above, you are setting the selected item in the combo box to be the 10th item (which happens to be the number 10). Indices in VB.Net begin with zero, not one. So the first item in the combo box has an index of zero, while the last item has an index of 9 in this case.

3. At the top of the code editor, click the Form1.vb [Design] tab.



This shows the form and the controls that you have placed on it. This is a convenient view for choosing a specific control to write code for. Now you'll add code to the button to compute average streamflow.

4. Double click the Calculate Average Streamflow button to open the code editor and automatically create stub code for the Click event for that button.
5. Add the following code to the btnCalculate_Click procedure.

```

' Set initial parameters.

' Set the siteCode for our gage of interest.
Dim location As String = "NWISUV:08158000"
' Set the variableCode for streamflow.
Dim variable As String = "NWISUV:00060"
' Set start and end date.
Dim startDate, endDate As String
Dim tmpDate As Date
endDate = Format(Now, "yyyy-MM-dd")
tmpDate = Now.AddDays(-1 * ComboBox1.SelectedItem + 1)
startDate = Format(tmpDate, "yyyy-MM-dd")

' Call the web service.

Dim ws As New NwisRealtime.WaterOneFlowSoapClient
Dim tsResponse As NwisRealtime.TimeSeriesResponseType
tsResponse = ws.GetValuesObject(location, variable, _
                                startDate, endDate, "")

' Process the results.

Dim vals As NwisRealtime.TsValuesSingleVariableType
vals = tsResponse.timeSeries.values
If vals.count = 0 Then
    MsgBox("No values returned")
    Exit Sub
End If

Dim avg As Double = 0

For i As Integer = 0 To vals.count - 1
    avg += vals.value(i).Value
Next

avg = avg / vals.count
MsgBox("The average streamflow is " & _
        FormatNumber(avg, 1) & " cfs")

```

In the code above, you are first preparing the inputs to feed the web service. The tricky part of this is formatting the dates to “yyyy-MM-dd” format (e.g., 2006-12-31), which is what the web service is expecting. Another trick is calculating the start date by adding “negative” days to the current date in the line:

```
tmpDate = Now.AddDays(-1 * ComboBox1.SelectedItem + 1)
```

Next you are creating a new instance of the NWIS Unit Values web service, and calling the `GetValuesObject` method from the service with the date inputs from the user. This method returns an Object with the data retrieved from the web service.

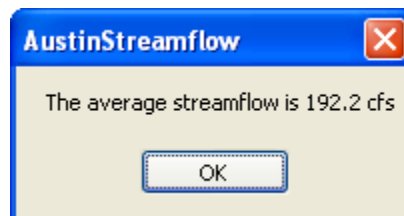
Next, with the results from the `GetValuesObject` call, you are computing the average streamflow from the values returned, and then showing a message box to report the result.

6.3.5 *Running the Code*

The project is now ready to run.

1. Press F5 on your keyboard to run it.
2. Click the Calculate Average Streamflow button.

After a minute or two, a message box appears showing the average streamflow over the past 10 days. Note that your value may be different than the value in the screenshot below, since this exercise was created on another day than the current day.



3. **Close** the form when you are finished.

You have the exercise and have learned how to call a web service from Visual Studio 2008. From this point, you could build the solution as an executable file by pressing Ctrl-Shift-B on your keyboard. See your Visual Studio help for more information about building solutions.

7.0 Ingesting NWIS Data Using Java

by David Valentine

In this chapter you will build a Java class that accesses the NWIS Daily Values web service to obtain daily streamflow values for Big Rock Creek near Valyermo, California, for the year 2001. The class will output the site code and site name for this location, as read from the web service, as well as the time series of streamflow values.

7.1 Computer and Skill Requirements

To complete this exercise, your computer must meet the following requirements:

- Working Internet connection
- Java SE 5: download from <http://java.sun.com/>
- NetBeans IDE 5.5: download from <http://www.netbeans.org>

This exercise assumes that you have some familiarity with general programming concepts and Java.

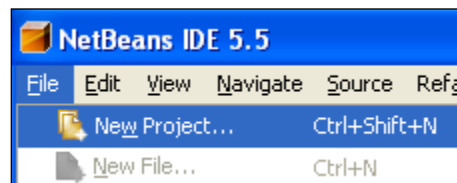
NOTE: The source code for the `nwis.java` class created in this exercise is located in Appendix C.

7.2 Procedure

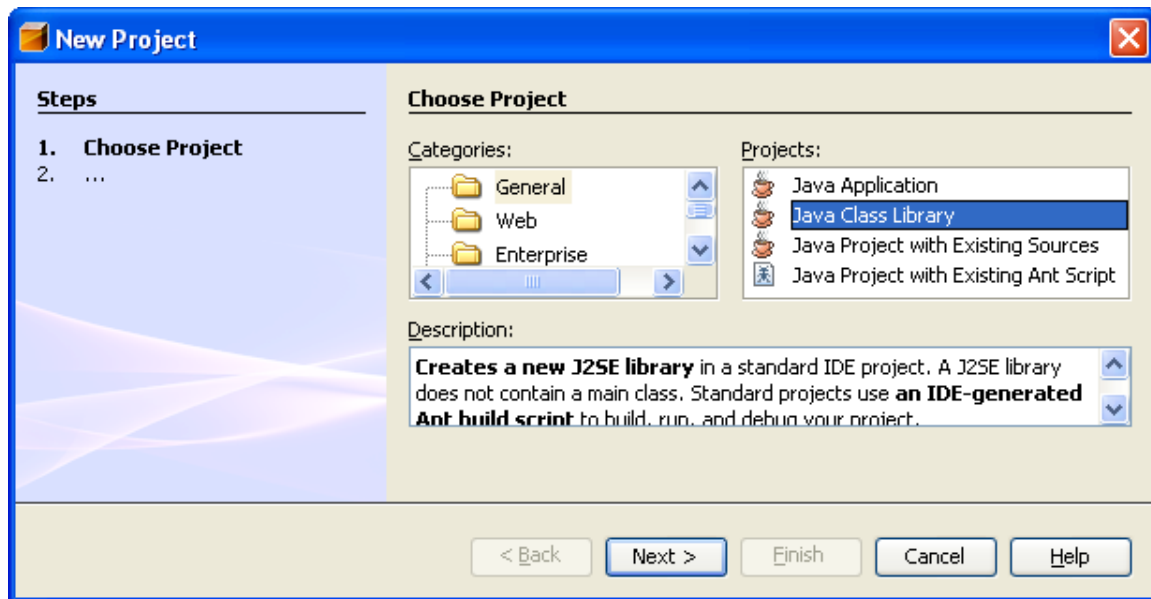
7.2.1 *Creating a New Project*

First, you will create a new Java project.

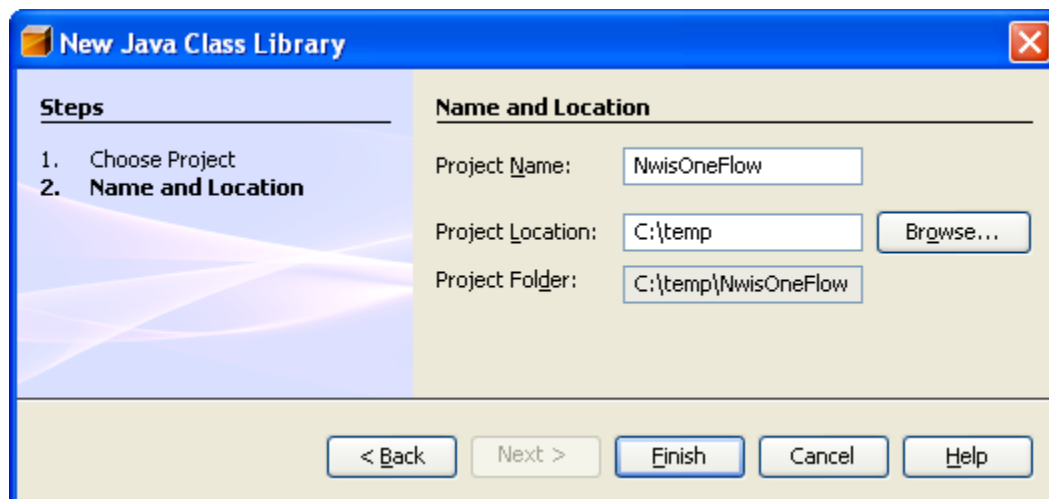
1. Start NetBeans IDE.
2. Click the File menu, and then click New Project...



3. In the New Project dialog, select General in the Categories pane. In the Projects pane, select Java Class Library.



4. Click Next.
5. In the New Java Class Library dialog, enter “NwisOneFlow” as the Project Name.
6. Enter the path in which you want the project folder to be created in Project Location. The IDE will create a subfolder at that location called “NwisOneFlow”, which will contain all files used in the project.

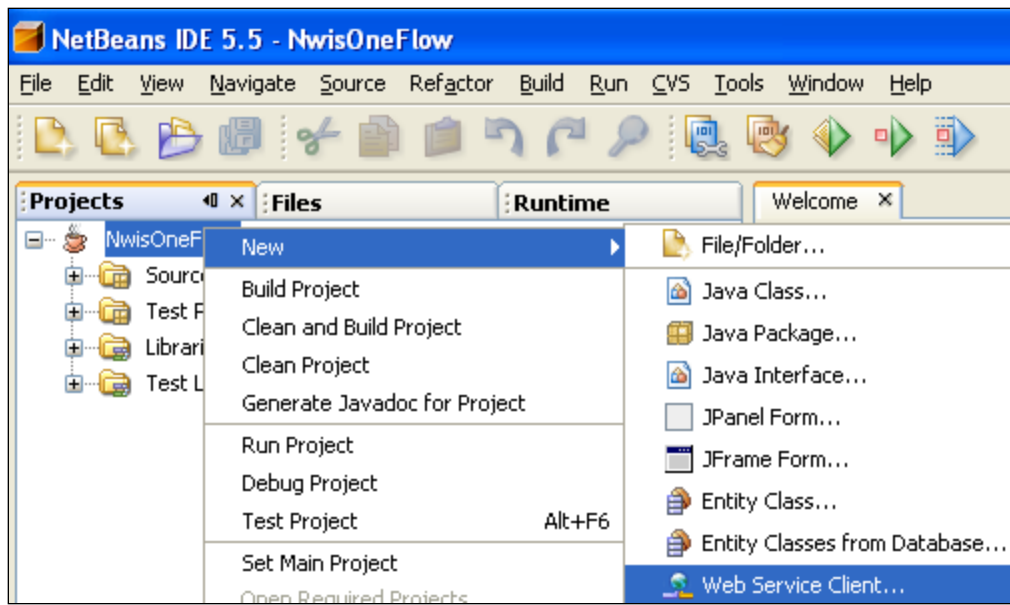


7. Click Finish.

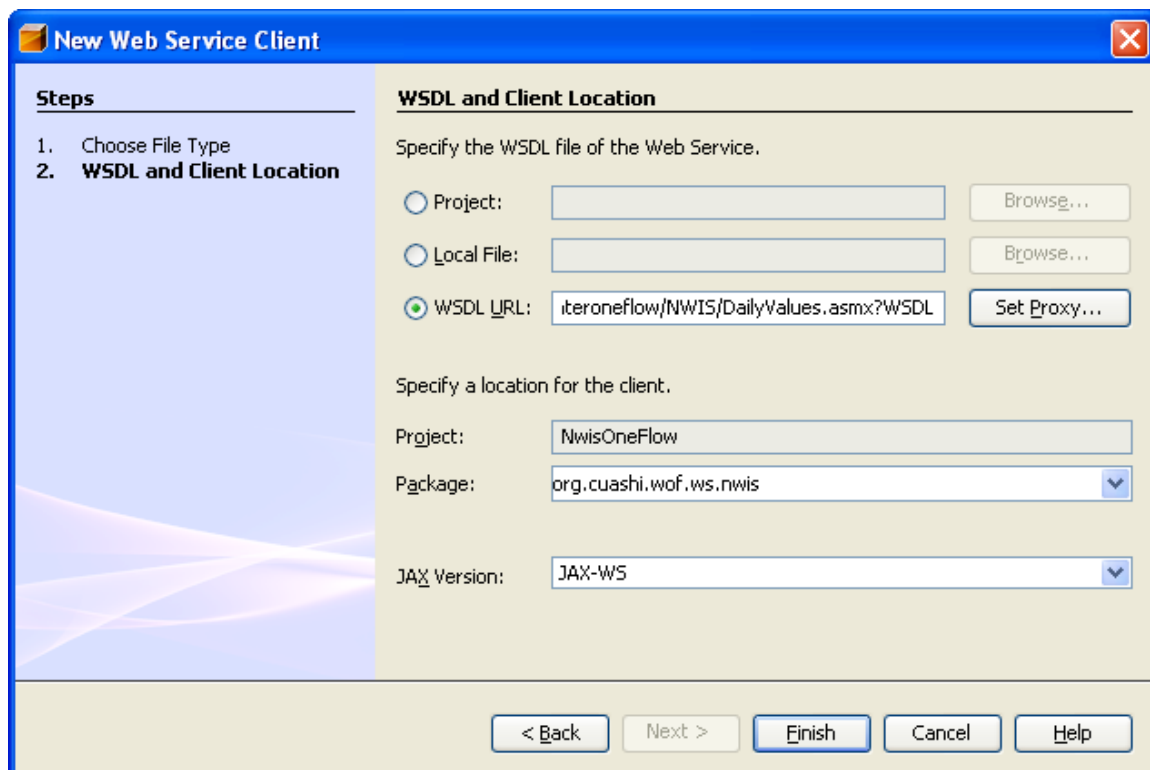
7.2.2 Creating a Web Service Client

With the project set up, you will now create a client for the NWIS web service.

1. In the Projects window, right click on NwisOneFlow, point to New, and then click Web Service Client...



2. In the New Web Service Client dialog, enter “http://water.sdsc.edu/wateroneflow/NWIS/DailyValues.asmx?WSDL” as the WSDL URL.
3. For the Package, enter “org.cuashi.wof.ws.nwis”
4. For the JAX Version, select JAX-WS.

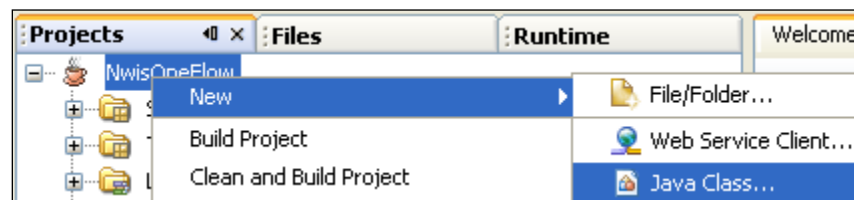


5. Click Finish to compile the web service client class.

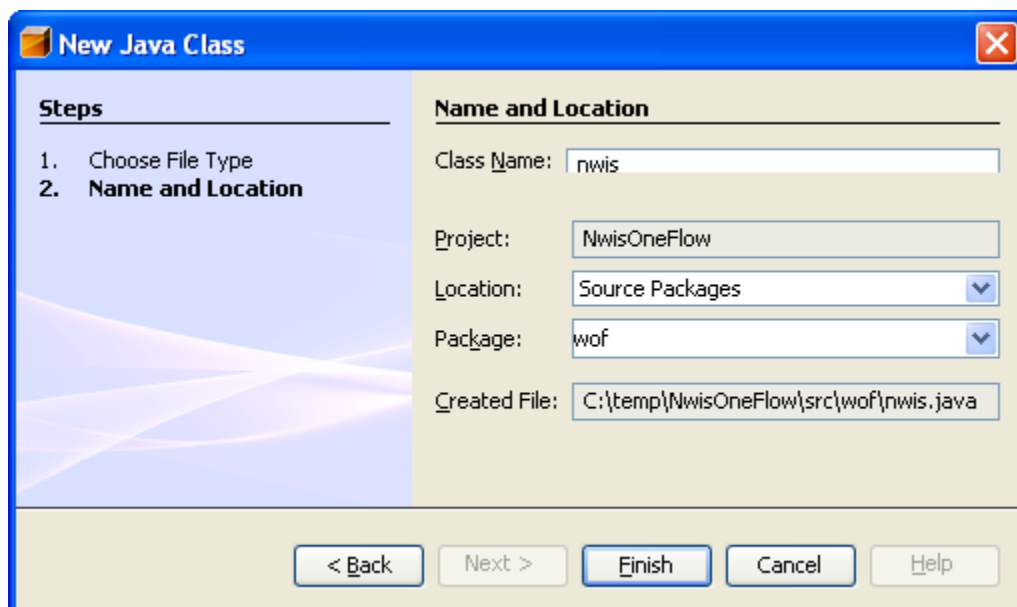
7.2.3 Creating a Class to Consume the Web Service

In this section you will create a new java class, wof.nwis, that accesses the web service client you just created. The class will download a time series of daily streamflow values at Big Rock Creek near Valyermo, California, for the year 2001. The site code for this location is 10263500, and the variable code for streamflow is 00060. You will hard code both of these values into the class. You will also hard code the time span (the year 2001). In a more robust application, you would let the user supply these parameters. The class will output the name of the site, its site code, and the time series of values.

1. In the Projects window, right click on the NwisOneFlow project, point to New, and then click Java Class...



2. In the New Java Class dialog, enter “nwis” as the Class Name, and “wof” as the *Package*.



3. Click Finish.

Now you will add a “main” procedure, which is the default procedure that will run when the class is invoked. It is within this procedure that you will eventually add the code to retrieve the time series values.

4. At the end of the source code for the nwis class, enter the following code.

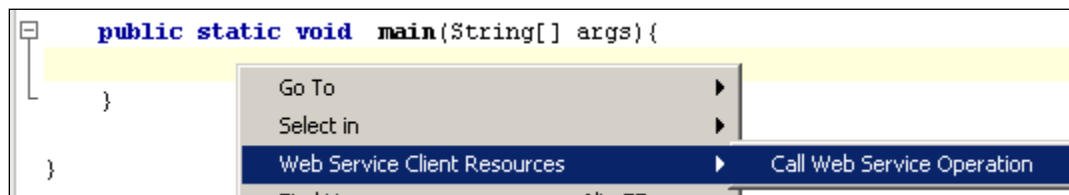
```
public static void main(String[] args){  
}
```

The screenshot below shows the result.

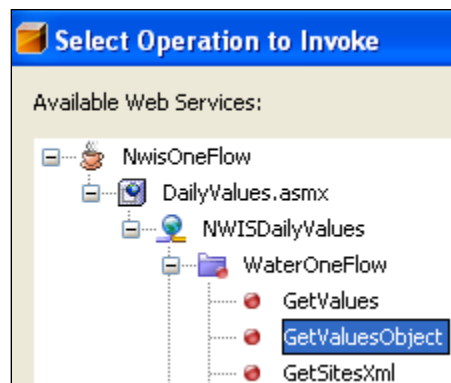
```
package wof;  
  
/**  
 *  
 * @author whiteaker  
 */  
public class nwis {  
  
    /** Creates a new instance of nwis */  
    public nwis() {  
    }  
  
    public static void main(String[] args){  
    }  
}
```

Now you will create the code for calling the web service. Fortunately, the IDE can create most of the code for you.

5. Right click within the code for the main method, point to Web Service Client Resources, and then click Call Web Service Operation.



6. In the Select Operations to Invoke dialog, select GetValuesObject, and click OK.



After you click OK, the IDE generates code for calling the `GetValuesObject` method from the NWIS web service. The IDE creates variables (e.g., `location`) for storing the parameters that will be sent to the web service, but leaves them empty for you to fill in later. A screenshot from the code editor is shown below.

```
public static void main(String[] args){
    try { // Call Web Service Operation
        org.cuashi.wof.ws.nwis.NWISDailyValues service =
            new org.cuashi.wof.ws.nwis.NWISDailyValues();
        org.cuashi.wof.ws.nwis.WaterOneFlow port =
            service.getWaterOneFlow();
        // TODO initialize WS operation arguments here
        java.lang.String location = "";
        java.lang.String variable = "";
        java.lang.String startDate = "";
        java.lang.String endDate = "";
        java.lang.String authToken = "";
        // TODO process result here
        org.cuashi.wof.ws.nwis.TimeSeriesResponseType result =
            port.getValuesObject(location, variable, startDate, endDate, authToken);
        System.out.println("Result = "+result);
    } catch (Exception ex) {
        // TODO handle custom exceptions here
    }
}
```

When executed, the above code creates a web service, gets an instance, and then calls `GetValuesObject`. Now you will hard code the parameters for our site of interest. Remember, you are hard coding these parameters for this simple example application, but a more robust application would read these parameters as inputs from the user.

7. Fill in the parameters for calling the web method.

```
java.lang.String location = "NWIS:10263500";
java.lang.String variable = "NIWS:00060";
java.lang.String startDate = "2001-01-01";
java.lang.String endDate = "2001-12-31";
java.lang.String authToken = "";
```

A screenshot from the code editor is shown below.

```

public static void main(String[] args){
    String siteCode = null;
    String siteName = null;

    try { // Call Web Service Operation
        org.cuashi.wof.ws.nwis.NWISDailyValues service =
            new org.cuashi.wof.ws.nwis.NWISDailyValues();
        org.cuashi.wof.ws.nwis.WaterOneFlow port =
            service.getWaterOneFlow();
        // TODO initialize WS operation arguments here
        java.lang.String location = "NWIS:10263500";
        java.lang.String variable = "NIWS:00060";
        java.lang.String startDate = "2001-01-01";
        java.lang.String endDate = "2001-12-31";
        java.lang.String authToken = "";
        // TODO process result here
        org.cuashi.wof.ws.nwis.TimeSeriesResponseType result =
            port.getValuesObject(location, variable, startDate, endDate, authToken);
        System.out.println("Result = "+result);
    } catch (Exception ex) {
        // TODO handle custom exceptions here
    }
}

```

Now you will create variables to store the site code and name as read from the web service.

8. In the main procedure, above the try statement, add the following lines of code.

```

String siteCode = null;
String siteName = null;

```

A screenshot of the code editor is shown below.

```

public static void main(String[] args){
    String siteCode = null;
    String siteName = null;

    try { // Call Web Service Operation

```

To output the datetimes and values in the time series, you will use a List object.

9. Below the package declaration, add an import statement for the List library.

```

import java.util.List;

```

A screenshot from the code editor is shown below.

```
package wof;

import java.util.List;

/**
```

You will now tell the IDE to output the site code and site name to the Output window of the IDE.

- At the end of the try statement, replace the line that begins with “System.out.println” with the following lines of code, in order to output the site information:

```
org.cuashi.wof.ws.nwis.SiteInfoType sit =
    (org.cuashi.wof.ws.nwis.SiteInfoType)
        result.getTimeSeries().getSourceInfo();
siteCode = sit.getSiteCode().get(0).getValue();
siteName = sit.getSiteName();

System.out.println("siteCode = "+siteCode);
System.out.println("siteName = "+siteName);
```

A screenshot from the code editor is shown below.

```
// TODO process result here
org.cuashi.wof.ws.nwis.TimeSeriesResponseType result =
    port.getValuesObject(location, variable, startDate, endDate, authToken);

org.cuashi.wof.ws.nwis.SiteInfoType sit =
    (org.cuashi.wof.ws.nwis.SiteInfoType) result.getTimeSeries().getSourceInfo();
siteCode = sit.getSiteCode().get(0).getValue();
siteName = sit.getSiteName();

System.out.println("siteCode = "+siteCode);
System.out.println("siteName = "+siteName);

} catch (Exception ex) {
    // TODO handle custom exceptions here
}
```

Some notes on the above code logic: You are working with objects, so you need to do some type casting in order to get the correct object. The line below takes a sourceInfo type and casts it to a siteInfo type.

```
org.cuashi.wof.ws.nwis.SiteInfoType sit =
    (org.cuashi.wof.ws.nwis.SiteInfoType)
        result.getTimeSeries().getSourceInfo();
```

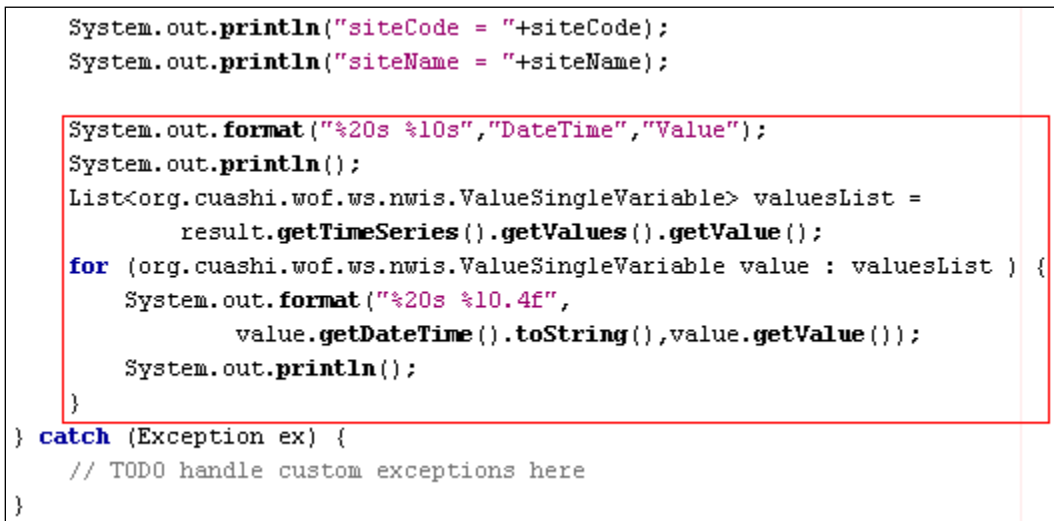
At present, there are two possible sourceInfo types: siteInfoType, and dataSetInfoType. If we were writing a more complete generic parser, we would use getClass().getName(), and cast based on the object type.

Finally, you will add code to output the time series values.

11. Add the flowing code after the last “System.out.println” line that you just added.

```
System.out.format("%20s %10s", "DateTime", "Value");
System.out.println();
List<org.cuashi.wof.ws.nwis.ValueSingleVariable> valuesList =
    result.getTimeSeries().getValues().getValue();
for (org.cuashi.wof.ws.nwis.ValueSingleVariable value : valuesList ) {
    System.out.format("%20s %10.4f",
        value.getDateTime().toString(), value.getValue());
    System.out.println();
}
```

A screenshot from the code editor is shown below.



```
System.out.println("siteCode = "+siteCode);
System.out.println("siteName = "+siteName);

System.out.format("%20s %10s", "DateTime", "Value");
System.out.println();
List<org.cuashi.wof.ws.nwis.ValueSingleVariable> valuesList =
    result.getTimeSeries().getValues().getValue();
for (org.cuashi.wof.ws.nwis.ValueSingleVariable value : valuesList ) {
    System.out.format("%20s %10.4f",
        value.getDateTime().toString(), value.getValue());
    System.out.println();
}
} catch (Exception ex) {
    // TODO handle custom exceptions here
}
```

In the above code, we use a List and a for loop, which are features of java 1.5 and above. We loop through the set of values, and output formatted strings.

When finished, the code for the main method should look as follows. Note that text for long lines is wrapped.

```
public static void main(String[] args){
    String siteCode = null;
    String siteName = null;

    try { // Call Web Service Operation
        org.cuashi.wof.ws.nwis.NWISDailyValues service =
            new org.cuashi.wof.ws.nwis.NWISDailyValues();
        org.cuashi.wof.ws.nwis.WaterOneFlow port =
            service.getWaterOneFlow();
        // TODO initialize WS operation arguments here
        java.lang.String location = "NWIS:10263500";
        java.lang.String variable = "NIWS:00060";
        java.lang.String startDate = "2001-01-01";
        java.lang.String endDate = "2001-12-31";
        java.lang.String authToken = "";
    }
```

```

        // TODO process result here
        org.cuashi.wof.ws.nwis.TimeSeriesResponseType result =
            port.getValuesObject(location, variable, startDate,
endDate, authToken);

        org.cuashi.wof.ws.nwis.SiteInfoType sit =
            (org.cuashi.wof.ws.nwis.SiteInfoType)
result.getTimeSeries().getSourceInfo();
        siteCode = sit.getSiteCode().get(0).getValue();
        siteName = sit.getSiteName();

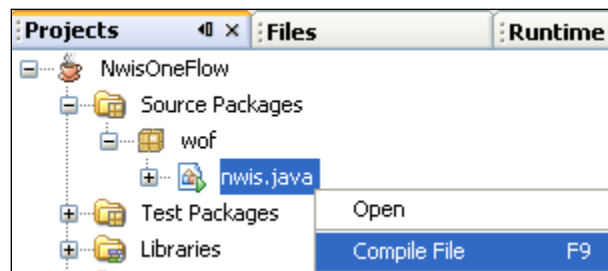
        System.out.println("siteCode = "+siteCode);
        System.out.println("siteName = "+siteName);

        System.out.format("%20s %10s", "DateTime", "Value");
        System.out.println();
        List<org.cuashi.wof.ws.nwis.ValueSingleVariable> valuesList =
            result.getTimeSeries().getValues().getValue();
        for (org.cuashi.wof.ws.nwis.ValueSingleVariable value :
valuesList ) {
            System.out.format("%20s %10.4f",
                value.getDateTime().toString(), value.getValue());
            System.out.println();
        }
    } catch (Exception ex) {
        // TODO handle custom exceptions here
    }
}

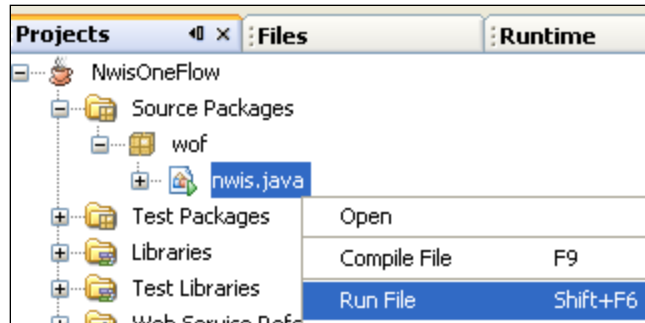
```

With the code finished, all that is left is to compile and run the file.

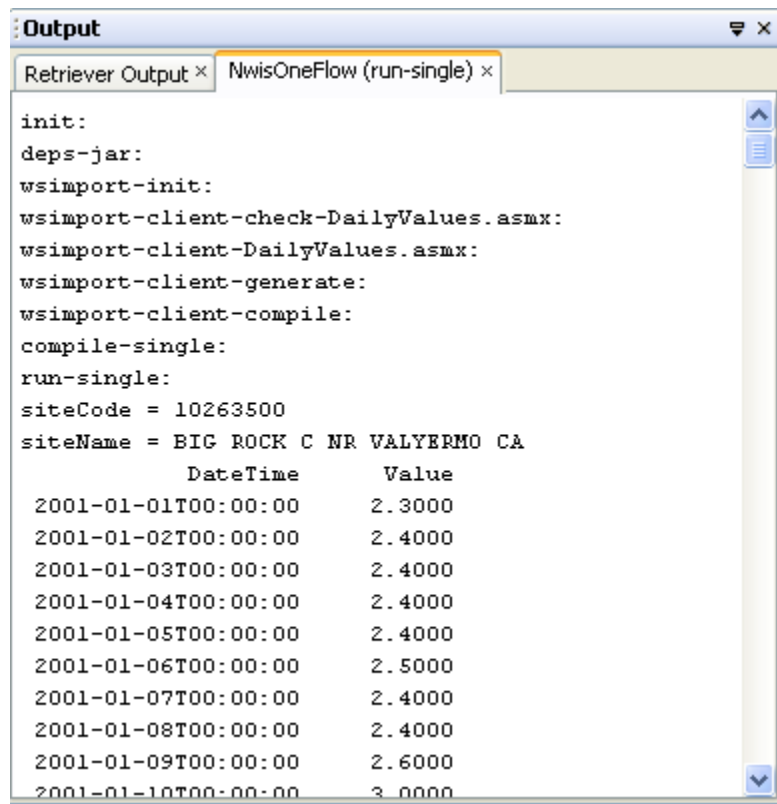
12. In the Projects window, right click on nwis.java and then click Compile File.



13. In the Projects window, right click on nwis.java and then click Run File.



In a moment, you will see the results of the `GetValuesObject` call as text in the Output window.



Congratulations! You have created a Java class which calls the NWIS web service to retrieve time series data. This concludes the exercise.

Appendix A: Source Code for parse_xml.m

```
% Function extracted from xmltools.m so that it could be called directly
% David Tarboton 3/19/06
% xmltools.m originally Matlab File Exchange
%
http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=3074
%
function [z, str] = parse_xml( str, current_tag, current_value, attribs, idx)

next = 'child';

if nargin < 2
    current_tag    = '';
    current_value  = '';
    attribs        = '';
    idx            = 0;
end
z = [];

eot = 0;

while ~eot & ~isempty(udeblank(deblank(str)))

    f_end = strfind(str, '</');
    f_beg = strfind(str, '<');

    %< Si je n'ai plus de tag dans mon document
    if isempty(f_end) & isempty(f_beg)

        if ~strcmp(lower(current_tag), '?xml') & ~isempty(current_tag)
            error('xmltools:parse_xml', 'malformed xml string (current [%s])',
current_tag);
        else
            fprintf('end parsing at level %d\n',idx);
            eot = 1;
            return
        end
    end
    %>

    if isempty(f_end)
        f_end = length(str)
    else
        f_end = f_end(1);
    end
    if isempty(f_beg)
        f_beg = length(str)
    else
        f_beg = f_beg(1);
    end

    if f_end <= f_beg
        %< je rencontre une fermeture
```

```

new_tag = str((f_end+2):end);
str_t   = str(1:f_end-1);
f_end = strfind(new_tag, '>');
if isempty(f_end)
    error('xmltools:parse_xml', 'malformed xml string : never ending tag
[%s] encountered', current_tag);
end
f_end = f_end(1);
str    = new_tag(f_end+1:end); % reste
new_tag = new_tag(1:f_end-1);
if ~strcmp(upper(new_tag), upper(current_tag))
    error('xmltools:parse_xml', 'malformed xml string : [%s] not properly
closed (closing [%s] encountered)', current_tag, new_tag);
end
% fprintf('%sclose [%s]\n', repmat(' ', 2*(idx-1),1), current_tag);
z.tag      = upper(current_tag);
z.attrs    = parse_attrs(attrs);
z.value    = udeblank(deblank(sprintf('%s %s', current_value, str_t)));
eot        = 1;
%>
else
    %< je rencontre une ouverture
    % je vais appeler le même code sur ce qu'il y a après moi
    current_value = sprintf('%s %s', current_value, str(1:f_beg-1));
    new_tag      = str(f_beg+1:end);
    f_end = strfind(new_tag, '>');
    if isempty(f_end)
        error('xmltools:parse_xml', 'malformed xml string : never ending tag
encountered');
    end
    f_end = f_end(1);
    str_t = new_tag(f_end+1:end);
    new_tag = new_tag(1:f_end-1);
    if (new_tag(end) == '/') | (new_tag(end) == '?')
        %< Self closing tag
        % Je met (temporairement!) eot à 1, cela me permet de passer quelques
lignes
        % de code tranquillement
        eot = 1;
        %>
    end
    %< Attributs
    f_beg = strfind(new_tag, ' ');
    if isempty(f_beg)
        new_attrs = '';
        if eot
            new_tag = new_tag(1:end-1);
        end
    else
        new_attrs = new_tag(f_beg+1:end);
        if eot
            new_attrs = new_attrs(1:end-1);
        end
        new_tag      = new_tag(1:f_beg-1);
    end
    %>
    % fprintf('%sopen [%s]\n', repmat(' ', 2*idx,1), new_tag);

```

```

    if eot
        %< If self-closing tag
        % fprintf('%s\nclose [%s]\n', repmat(' ', 2*idx,1), new_tag);
        new_attribs = parse_attribs( new_attribs);
        if isfield(z, next)
            nxt = getfield(z, next);
            nxt(end+1) = struct( 'tag', new_tag, 'attribs', new_attribs, 'value',
'', next, []);
            z = setfield(z, next, nxt);
            %z.(next)(end+1) = struct( 'tag', new_tag, 'attribs', new_attribs,
'value', '', next, []);
        else
            z = setfield(z, next, struct( 'tag', new_tag, 'attribs', new_attribs,
'value', '', next, [] ));
            %z.(next) = struct( 'tag', new_tag, 'attribs', new_attribs, 'value', '',
next, []);
        end
        str = str_t;
        eot = 0;
        %>
    else
        %< Appel du même code sur la suite

        % et stockage du resultat dans mes children.
        % Le code met aussi à jour le string courant |str|,
        % il en enlève la partie correspondant au string que je viens de
trouver.
        [t,str] = parse_xml(str_t, new_tag, '', new_attribs, 1+idx);
        if isfield(t, next)
            nx = getfield( t, next);
            %nx = t.(next);
        else
            nx = [];
        end
        if isfield(z, next)
            nxt = getfield(z, next);
            nxt(end+1) = struct( 'tag', t.tag, 'attribs', t.attribs, 'value',
t.value, next, nx);
            z = setfield(z, next, nxt);
            %z.(next)(end+1) = struct( 'tag', t.tag, 'attribs', t.attribs, 'value',
t.value, next, nx);
        else
            z = setfield(z, next, struct( 'tag', t.tag, 'attribs', t.attribs, 'value',
t.value, next, nx));
            %z.(next) = struct( 'tag', t.tag, 'attribs', t.attribs, 'value', t.value,
next, nx);
        end

        %>
    end
end
end
%>
end
%>

```

```

%< Parse attribs
function z = parse_attribs( a)
if isempty(a)
    z = struct( 'name', '', 'value', '');
    return
end
b = tokens(a, ' ');
j = 1;
for i=1:length(b)
    if ~isempty(b{i})
        t = tokens(b{i}, '=');
        if length(t)==2
            u = t{2};
            if u(1)=='"'
                u = u(2:end);
            end
            if u(end)=='"'
                u = u(1:end-1);
            end
            z(j) = struct( 'name', upper(t{1}), 'value', u);
        else
            z(j) = struct( 'name', upper(a), 'value', '');
        end
        j = j +1;
    end
end
%>

%<* Ecriture d'une structure xml
function z = write_xml(fid, xml_struct, idx)

next = 'child';

if nargin < 3
    idx = 0;
end

margin = repmat(' ',2*idx,1);

closed_tag = 1;
%< Ouverture du tag
if isfield(xml_struct, 'tag')
    closed_tag = 0;
    fprintf(fid, '%s<%s', margin, xml_struct.tag);
    %< Ecriture des attributs
    if ~isfield(xml_struct, 'attribs')
        error('xmltools:write_xml', 'malformed MATLAB xml structure : tag without attribs');
    end
    for i=1:length(xml_struct.attribs)
        if ~isempty(xml_struct.attribs(i).name)
            fprintf(fid, ' %s="%s"', xml_struct.attribs(i).name,
xml_struct.attribs(i).value);
        end
    end
end
%>

```

```

%< Gestion des Auto closed tags
% Si le tag n'est pas auto fermé, alors |closed_tag| est à zéro
if ~isfield(xml_struct, next)
    error('xmltools:write_xml', 'malformed MATLAB xml structure : tag
without %s', next);
end
if ~isfield(xml_struct, 'value')
    error('xmltools:write_xml', 'malformed MATLAB xml structure : tag without
value');
end
if xml_struct.tag(1) == '?'
    fprintf(fid, '?>\n');
    closed_tag = 1;
elseif isempty(getfield(xml_struct, next)) & isempty(xml_struct.value)
%elseif isempty(xml_struct.(next)) & isempty(xml_struct.value)
    fprintf(fid, '/>\n');
    closed_tag = 1;
else
    fprintf(fid, '>\n');
end
%>
end
%>

%< Ecriture de la value
if isfield(xml_struct, 'value')
    if ~isempty(xml_struct.value)
        fprintf(fid, '%s%s\n', margin, xml_struct.value);
    end
end
%>

%< Ecriture des enfants
if ~isfield(xml_struct, next)
    error('xmltools:write_xml', 'malformed MATLAB xml structure : tag
without %s', next);
end
those_children = getfield(xml_struct, next);
%those_children = xml_struct.(next);
for i=1:length(those_children)
    write_xml(fid, those_children(i), idx+1);
end
%>

%< Fermeture du tag
if ~closed_tag
    fprintf(fid, '%s</%s>\n', margin, xml_struct.tag);
end
%>
%>*

%<* get childs with a specific tag name
function z = get_childs(z, next, tag_name);
u = getfield(z, next);

```



```

zo = [];
for i=1:length(u)
    v = u(i);
    if strcmp(upper(v.tag), upper(tag_name))
        if isempty(zo)
            zo.anext= v;
        else
            zo.anext(end+1) = v;
        end
    end
end
if ~isstruct( zo)
    if isfield(z, 'tag')
        tn = z.tag;
    else
        tn = 'root?';
    end
    error('XMLTOOLS:GET-TEG', 'problem in finding tag <%s> under one <%s>',
tag_name, tn);
end
z = [ zo.anext ];
%>*

%< udeblank
function s = udeblank(str)
s = deblank(str(end:-1:1));
s = s(end:-1:1);
if length(s)==0
    s = '';
end
%>

%< emptystruct
function z = emptystruct(next)
z = struct( 'tag', [], 'value', [], 'attribs', [], next, []);
%>

%< Tokens
function l = tokens(str,del)
l={} ;
% Boucle sur les tokens.
del = sprintf(del) ;
while ~isempty(str)
    [tok,str] = strtok(str,del) ;
    l{end+1} = tok ;
end
%>

```

Appendix B: Source Code for MODISPlot_xml.m

```
% Initialize variables.
clear values
clear dn

% Create class.
wsdl='http://water.sdsc.edu/waterOneFlow/MODIS/Service.asmx?WSDL';
createClassFromWsd1(wsdl);

% This creates an instance of the class.
svsMODIS = MODIS;

% Specify input parameters.
w='-98.2' % West longitude.
s='30' % South latitude.
e='-97.3' % East longitude.
n='30.7' % North latitude.
location=['GEOM:BOX(','w',' ','s',' ','e',' ','n','')']
% Variable Code 11 = Cloud Optical Thickness Water Phase.
variableCode='MODIS:11/plotarea=land'
startDate='2004-01-01'
endDate='2004-12-01'

% Call the GetValues function to get the time series data.
xmlValues=GetValues(svsMODIS,location,variableCode, ...
    startDate,endDate, '')

% Parse the XML string.
structValues=parse_xml(xmlValues);

% Report the name and units of the chosen variable.
display(structValues.child.child(2).child(2).child(2).value)
display(structValues.child.child(2).child(2).child(3).value)

% Get the <value> tags.
Recs=structValues.child.child(2).child(3).child;
[d1,d2]=size(Recs)

% Build arrays of datetimes and values.
for i=1:d2
    % Reformat date to that Matlab can understand it.
    datetime=Recs(i).attrs(1).value;
    year=datetime(1:4);
    month=datetime(6:7);
    day=datetime(9:10);
    datetime=[month,'/',day,'/',year];
    dn(i)=datenum(datetime); % Convert to numeric date.
    % Read the time series value.
    values(i)=str2double(Recs(i).value);
end
```

```
% Plot the graph.  
plot(dn,values);datetick;  
grid on % Turn on grid lines for this plot.
```

Appendix C: Source Code for nwis.java Class

```
/*
 * nwis.java
 *
 * Created on November 10, 2006, 1:15 PM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

package wof;

import java.util.List;

public class nwis {

    /** Creates a new instance of nwis */
    public nwis() {
    }

    public static void main(String[] args){
        String siteCode = null;
        String siteName = null;

        try { // Call Web Service Operation
            org.cuashi.wof.ws.nwis.NWISDailyValues service =
                new org.cuashi.wof.ws.nwis.NWISDailyValues();
            org.cuashi.wof.ws.nwis.WaterOneFlow port =
                service.getWaterOneFlow();
            // TODO initialize WS operation arguments here
            java.lang.String location = "NWIS:10263500";
            java.lang.String variable = "NIWS:00060";
            java.lang.String startDate = "2001-01-01";
            java.lang.String endDate = "2001-12-31";
            java.lang.String authToken = "";
            // TODO process result here
            org.cuashi.wof.ws.nwis.TimeSeriesResponseType result =
                port.getValuesObject(location, variable, startDate,
endDate, authToken);

            org.cuashi.wof.ws.nwis.SiteInfoType sit =
                (org.cuashi.wof.ws.nwis.SiteInfoType)
result.getTimeSeries().getSourceInfo();
            siteCode = sit.getSiteCode().get(0).getValue();
            siteName = sit.getSiteName();

            System.out.println("siteCode = "+siteCode);
            System.out.println("siteName = "+siteName);

            System.out.format("%20s %10s", "DateTime", "Value");
            System.out.println();
            List<org.cuashi.wof.ws.nwis.ValueSingleVariable> valuesList =
                result.getTimeSeries().getValues().getValue();
            for (org.cuashi.wof.ws.nwis.ValueSingleVariable value :
valuesList ) {
```

```
        System.out.format("%20s %10.4f",
            value.getDateTime().toString(),value.getValue());
        System.out.println();
    }
} catch (Exception ex) {
    // TODO handle custom exceptions here
}

}

}
```