A Beginners Guide to APL2000 Web Services

Brian Chizever

2005 APL2000 User Conference

Overview

Web Services is a way of exposing applications and data over the web in a standard way. This paper will focus on some of the basics of APL2000 Web Services and how to use it to handle requests from a web browser.

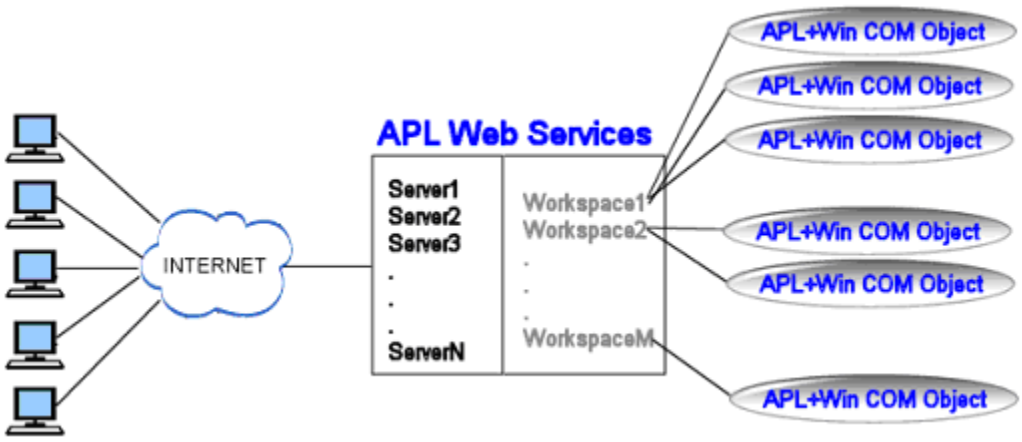
What is APL Web Services?

APL Web Services is APL2000's means of exposing Web Services to the APL programmer. (This may be referred to as AplServices or APL+WebServices in some contexts.) It is an NT service which means that it can run in the background, even if you are not logged in, and generally does not interact with the user.

APL Web Services is also a container. It manages and configures multiple web servers. It also manages APL+Win COM objects which will load workspaces. And finally it ties requests to its web servers to functions within these managed workspaces. (Note that a single machine can only run a single instance of APL Web Services, but that APL Web Services can run multiple web servers.)

A Picture Is Worth A Thousand Words

- a user types a request into an internet browser
- the request goes over the internet to its destination server which may be in APL Web Services
- APL Web Services handles the request by doing one of the following:
  - return a static page
  - call a function in one of its workspaces
    - create an APL+Win COM object (if needed)
    - load the workspace (if the COM object was just created)
    - call the function with any arguments specified in the configuration
    - get the function result and return it to the browser based on any settings in the configuration
  - some other operation



How Does This Differ From []NI?

[]NI, a fairly thin cover for Winsock, was released in version 3.0 of APL+Win. A beta was distributed at the 1997 User Conference. []NI is much more generalized than APL Web Services, but also requires more effort on the part of the APL programmer.

APL Web Services	**[]NI**
high level	low level
runs as a service	does not run as a service
automatically handles load balancing between multiple processes on a single machine	does no automatic load balancing
only handles web services	handles any protocol
server only	client and server

Web Servers

All web servers are uniquely identified by two attributes, an IP address and a port. Together, these determine the machine being queried, and what server on that machine.

IP Address

All computers accessible via the internet have a unique IP address. This address takes the form of four sets of numbers separated by dots. Each number is in the range from zero to 255. For example, Yahoo's web site has an IP address of 216.109.118.65. To make these addresses easier to remember, most web sites also get a friendly name, such as www.Yahoo.com. When you type a friendly name into your browser, the Domain Name Server (DNS) resolves the friendly name to the IP address. This IP address is then used to find the machine hosting that web site.

A single physical computer can have more than one IP address, but any single IP address specifies a single machine. You can think of it this way. A building might have only a single mailing address, which is the case in a single family home, or it could have multiple mailing addresses, such as in an apartment building. Similarly, a computer may have a single IP address, or it may have multiple IP addresses. (A machine with multiple IP addressed is called a "multi-homed host".) In the same way that you can determine the building from a mailing address, you can identify the computer from the IP address.

localhost

The friendly name *localhost*, which resolves to 127.0.0.1 is always the local machine. This is also known as the loopback adapter because is only accessible when the client and server are on the same machine. If you set up a server on 127.0.0.1, you can test it from a client on the same machine, but no other machines will be able to access that server.

When a server is used in a production environment, an IP address that is accessible from other machines is required

Ports

A server communicates on a specific port. The port is specified as an integer between one and 65535. Port numbers less than 1024 are "well known" or standard ports. For example, most web servers use port 80 and most FTP servers use port 21. There is no requirement to use these numbers, but most web browsers default to port 80. If the web server were to use a different port, you would need to specify that port number when browsing to that server.

You cannot have two servers with the same IP address use the same port. When the first server starts it will bind to that port so, when the second one starts, the port will be unavailable and the second server will fail. For this reason, if you are running multiple web servers on a single machine, but that machine has only a single IP address, each web server will need to use its own port.

Home Directory

A home directory is a mapping from a web server to a directory on the host machine. Your domain name or IP address is mapped directly to this directory. The home directory is the top of your web publishing tree so paths specified in the web browser are sub-folders of the home directory.

Since a single computer may host more than one web site (using either multiple IP addresses and/or multiple ports), each web server gets its own home directory. For example, if you set up your computer with the following web servers:

IP Address	Port	Home Directory
10.10.10.1	80	C:\web\one
10.10.10.2	80	C:\web\two
10.10.10.2	8080	C:\web\three

A web request of http://10.10.10.1/page.htm will be interpreted by the web server as a request for the C:\web\one\page.htm file. A request of http://10.10.10.2/picture.gif will be interpreted by the web server as a request for the C:\web\two\picture.gif file. Finally a request for http://10.10.10.2:8080/photo.jpg is a request for the C:\web\three\photo.jpg file.

Note that in the third example we needed to specify the port. Internet Explorer, Firefox, Netscape Navigator, and most other web browsers will default to port 80 if not specified.

Virtual Directories

A virtual directory is a directory that exists outside of the home directory tree. For APL Web Services, instead of being a physical directory, it is instead a pointer to a function in a particular workspace. The virtual directory appears to client browsers as though it were a physical directory.

A virtual directory is an alias that does not reveal any information about either the workspace location or the real name of the function which it references.

Default Pages

If the browser makes a request but does not include a specific page, the web server will generally return a default page. For example, the Yahoo web site is configured to return the index.html file if no file is specified. That means that http://www.Yahoo.com and http://www.Yahoo.com/index.html, will both be interpreted by the web server as a request for the index.html file in its home directory.

Web Browsers

Web browsers do two things. They make requests of web servers and they display the response.

The simplest way to make a request is to type a URI (Uniform Resource Identifier) into the address field. The simplest form of this URI is www.apl2000.com. A more complete URI is http://IPaddress:port/path/file. The pieces of the request are as follows:

http	The protocol, or agreed upon language of communication. It stands for HyperText Transfer Protocol. Most web browsers will supply this if it is not specified.
IPaddress	The IP address or friendly name of the server.
port	The port the server is using. If omitted, it defaults to 80.
path	The subdirectory of the home directory to use. The path may also map to a function in an APL workspace.
file	The requested file. If not specified, the default page in that path, if it exists, is returned.

Installing APL Web Services

Before installing APL Web Services, make sure you have the correct system requirements and that APL+Win is installed

The minimum system requirements are Windows NT 4.0 SP6a or Windows 2000/XP/2003, Microsoft .NET Framework version 1.1, and IE 5.01 or greater.

You also need APL+Win version 5 or 6, and the APL+Win COM object installed. To ensure you have APL+Win correctly installed, follow these steps:

- Check the version of APL+Win by starting APL and looking at the result of []SYSVER.
- Check that APL+Win is a valid COM object by typing:
  ```
  '#' ⎕wi 'XInfo' 'APLW'
  ```
  If the result is not empty, APL+Win is a COM object.
- Check the version of the APL+Win COM object by creating an instance of it and checking its []SYSVER variable:
  ```
  'x' ⎕wi 'New' 'APLW.WSEngine'
  'x' ⎕wi 'SysVariable' 'SYSVER'
  ```
- Determine if the APL+Win COM object is runtime or not by looking at the 21st element of the []SYS variable:
  ```
  ('x' ⎕wi 'SysVariable' 'SYS')[21]
  ```
  If the result is 1, the runtime is installed. If the result is 0, it's the non-runtime.

If any of the above checks fail, you can fix it by re-installed APL+Win. You can also register APL+Win as a COM object manually by registering the aplwCo.dll file

regsvr32.exe *d:\path*\aplwco.dll

In a production environment, you would want to use the runtime APL. For development, and for this session, you'll want to use the non-runtime version:

d:\path\aplw.exe *arguments* /RegServer

To install APL Web Services, simply double-click the AplServicesSetup.msi icon. The installation program will create the *C:\Program Files\APL2000\AplWebServices* directory and place icons on your desktop and in your Start menu for the APL Web Services Admin program, which will allow you to administer APL Web Services, and for the APL Web Services help. A second help file, AplWebServicesReference.chm, is also installed but, since it's a reference help file, a link to it is not put on your start menu.

It also creates the AplServices service. To see this, open the Windows Service Control Manager. Depending on the version of your operating system and how you have it configured, it may be found in different places. You can first try opening the Start menu and choosing Settings – Control Panel – Administrative Tools – Services. You should see a row with the name "AplServices" whose status is blank and whose Startup Type is "automatic". The startup type indicates that this service will start automatically every time you boot your computer. It's not yet started because we just installed it. Start it now by highlighting the row and pressing the start button in the toolbar.

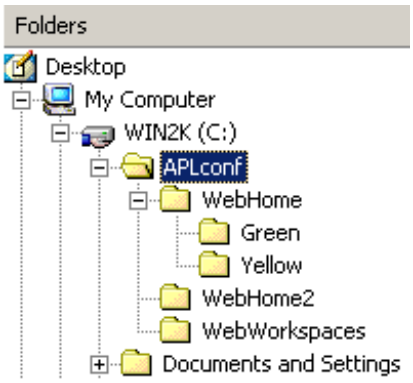
Configuring APL Web Services

Now that we have installed APL Web Services, we need to create our first server.

Home Directory

We first create a home directory to hold the static content for our web service. You can choose any location on your machine as your home directory. Remember that the home directory and its contents, including files and sub-folders, will be accessible to the internet. It will not be possible for anyone on the internet to access any other location on your hard drives using APL Web Services other the home directory.

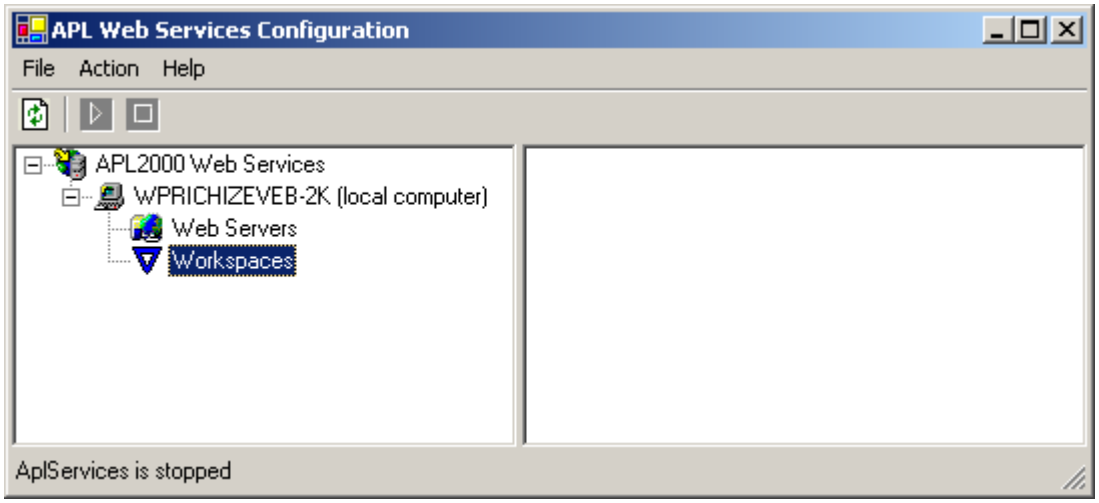
All the files needed in this paper are found in the BeginnerWebServices.exe self-extracting zip file. To follow the examples in this paper, unzip the files and subfolders to the root of your C-drive so that it looks like this:



Adding a Web Server

Start *APL Web Services Admin* by double-clicking its icon on the desktop or in the Start menu.

Add a new web server by right-clicking the "Web Servers" node and choosing New Server. This will create a new item called *defaultwebsite*.

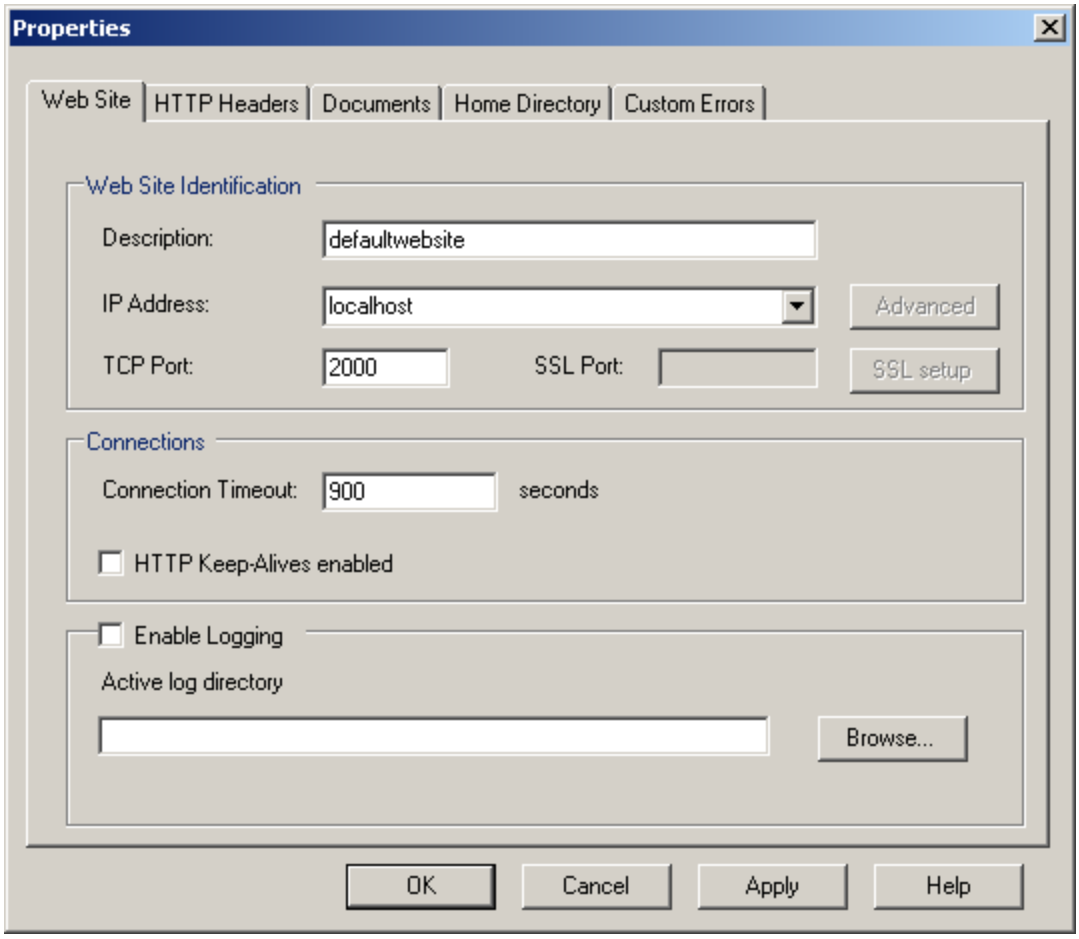


Web Server Properties

Set the properties of this server by right-clicking it and choosing Properties.

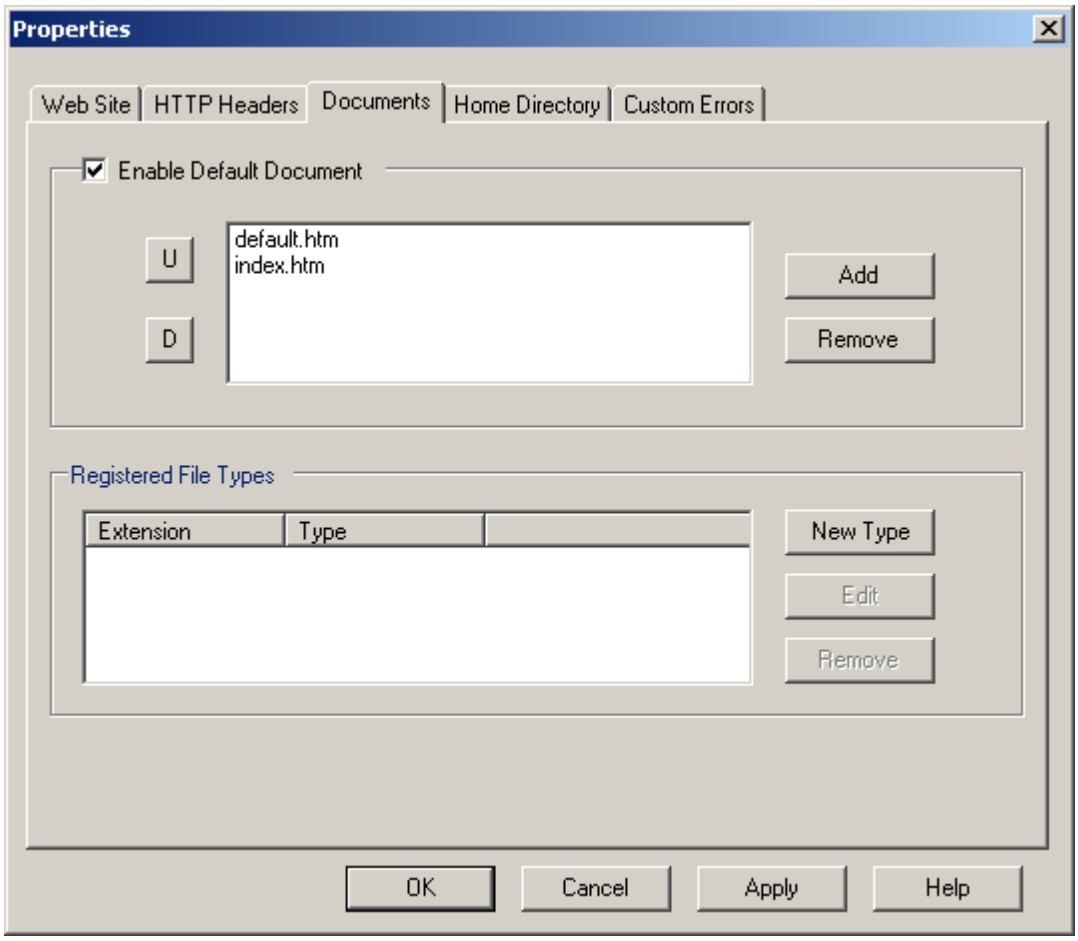
Web Site

The Web Site tab is where you specify the IP address and port. We will use localhost (the local machine) because the server and browser are on the same machine. Remember that in a production environment, you would need to use an actual IP address. We will use port 2000 because it is above the range of well known ports and because it is unlikely to be in use.



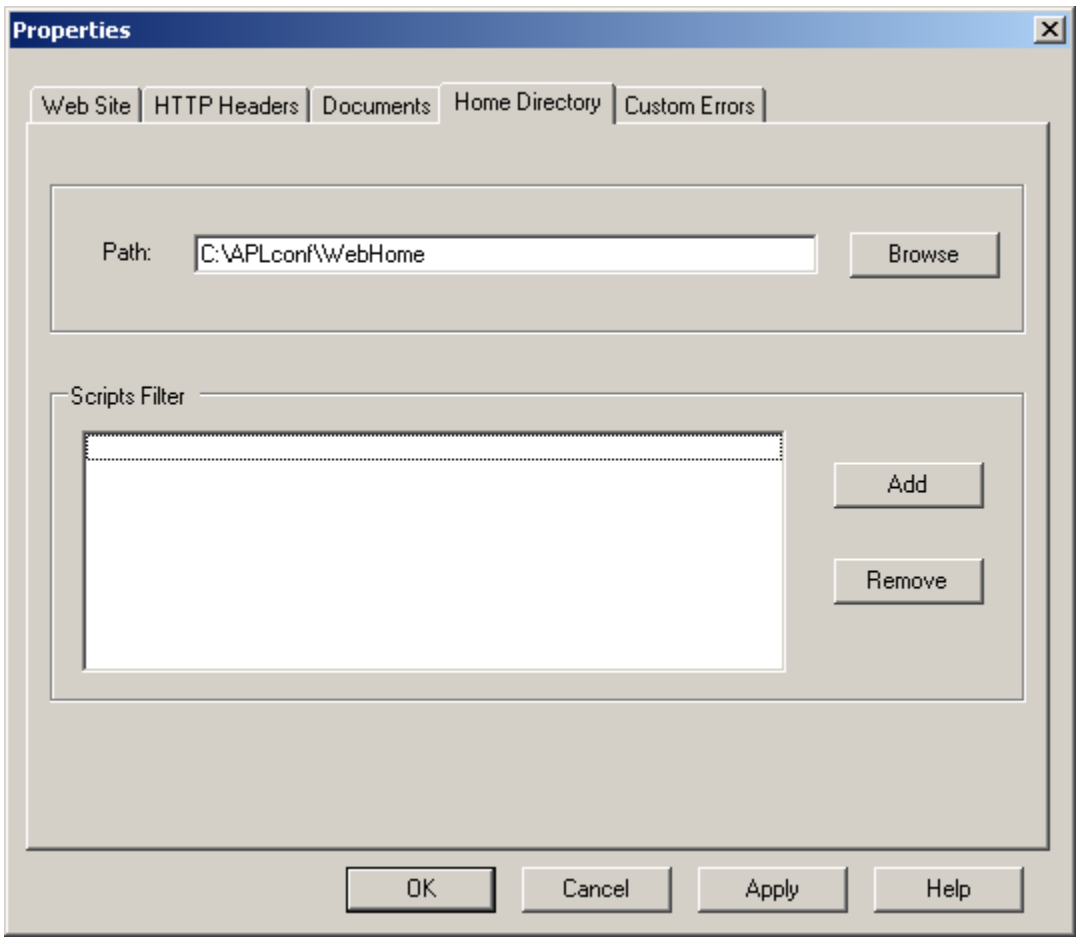
Documents

The Documents tab lets us specify if our server will return a default page, and if so, what file name to use. Check Enable Default Documents and put default.htm in the list. Note that you can list multiple files and the server will search for each file in the order listed until one is found.



Home Directory

The Home Directory tab is where you specify the location on the disk drive from which the server should retrieve static contents. Enter the path to the WebHome directory which we created earlier.



You then click OK to save these settings.

Starting and Testing the Server

We already started APL Web Services from the Windows Service Control Manager. We now need to start the web server, *defaultwebsite*, that we just created. To do this, highlight *defaultwebsite* and start it using the Start menu on the Action menu.

Open a web browser and try the following addresses:

http://localhost:2000	You should see the default web page in the home directory.
http://localhost:2000/static.htm	You should see a static page in the home directory.
http://localhost:2000/yellow/static.htm	You should see a static page in the Yellow subfolder of the home directory.
http://localhost:2000/yellow/	You should see the default page in the Yellow subfolder of the home directory.
http://localhost:2000/yellow	You should get an error. Since there's no trailing slash, it's looking for the "yellow" page in the home folder, which doesn't exist.
http://localhost:2000/green/static.htm	You should see a static page in the Green subfolder of the home directory.
http://localhost:2000/green/	You should see an error because there is no default page in the green subfolder of the home directory.
http://localhost:2000/green	You should see an error because there is no green page in the home directory

Using APL to Return Content

So far all we've done is show that we can use APL Web Services to serve up static pages. This could have been done using any number of free web servers. What we want to do is have APL Web Services call a function in an APL workspace, instead of just returning a file from the hard drive.

Adding a Workspace

In APL Web Services Configuration, right-click on the Workspaces node and select New Workspace to add a new node to the tree called *defaultworkspace*. Right-click on *defaultworkspace*, select Rename and type "basic1". This is the name by which we will reference this workspace.

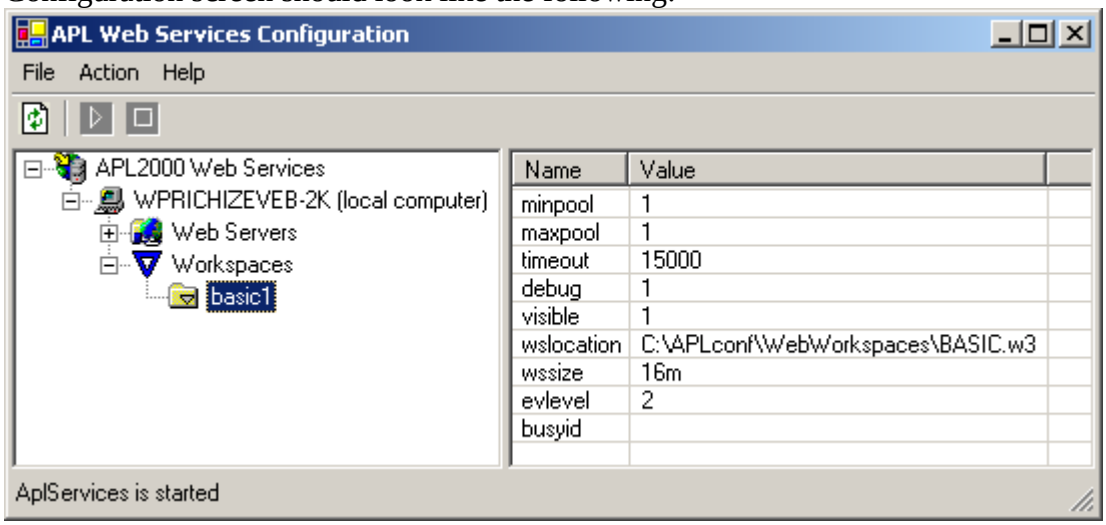
Setting Workspace Properties

The most important property to set is *wslocation*. Right-click on *wslocation* in the listview, select modify, and type in the full path and name (or browse to) the workspace you wish to use. For our examples we'll use the BASIC.w3 workspace in the C:\APLConf\WebWorkspaces directory.

The other properties that we'll set at this time are *debug* and *visible*. Remember that APL Web Services will start an APL+Win COM object and then load the specified workspace into it. The APL+Win COM object is generally hidden, but when developing, it's often helpful to see the APL+Win COM object so that you can modify, trace, and debug the functions in the running workspace.

If the *visible* property is set to one, the APL+Win COM object and workspace will be shown. If *visible* is zero, but *debug* is one, then the COM object starts out hidden, but will be made visible if a function suspends due to error or a []STOP in the function.

After setting the *wslocation*, *debug*, and *visible* properties, the APL Web Services Configuration screen should look like the following:



Some of the other properties, such as *minpool* and *maxpool*, will be discussed later in this paper.

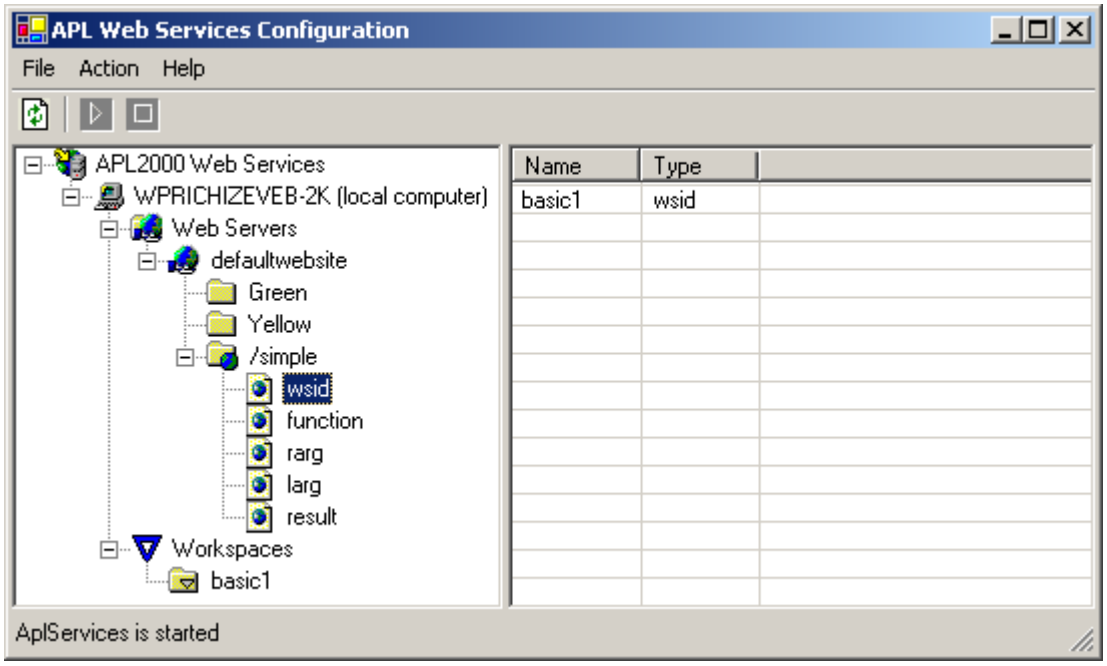
Right-click on "basic1" and choose "Start" to make this workspace available.

Connecting a URI to an APL Function

We now need to tell the web server that some requests should call functions in that workspace instead of simply looking for a specific file. Right-click *defaultwebsite* and choose New Virtual Directory. You should now see "/default" appear under *defaultwebsite*. Right-click it, choose Rename, and type "/simple".

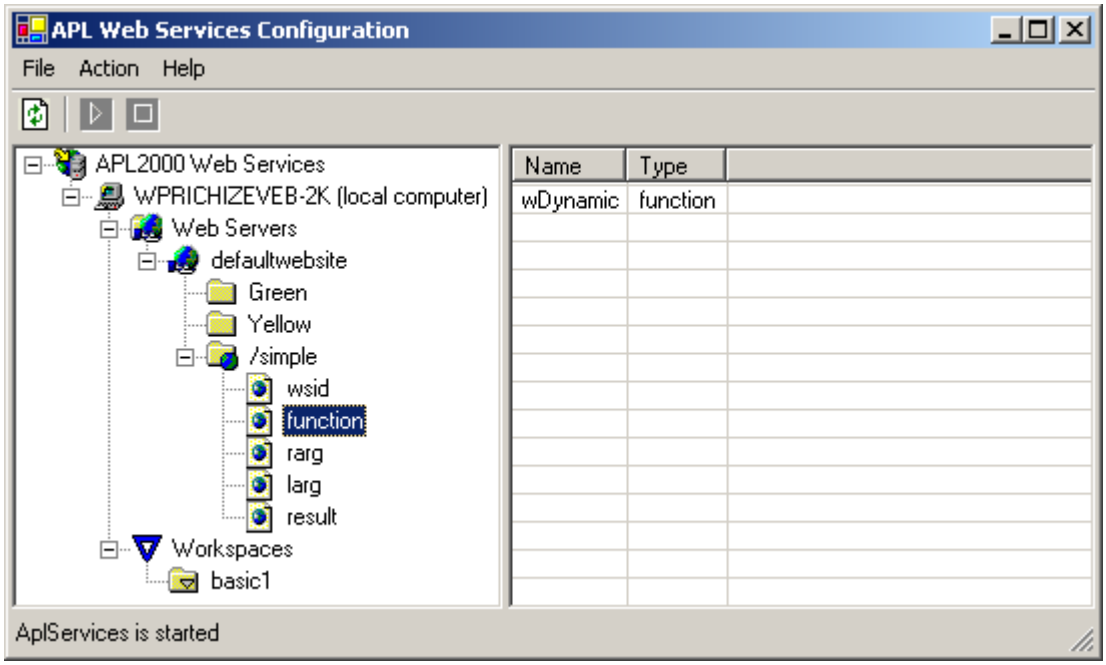
wsid

Click on the wsid node in the tree, then right-click the top row in the listview, choose Modify, and set the name to "basic1". This is the name we used in the previous step when adding the workspace



function

Click the function node of the tree, right-click the top row of the listview, choose Modify, and type “wDynamic” as the name.



wDynamic is the name of a function in the BASIC.w3 workspace. It takes no arguments and returns a valid html document so there is no need to modify the rarg, larg, or result.

```
    ∇ z←wDynamic
[1]   ⍝ create a simple dynamic page
[2]   ⍝ return a complete html page
[3]
[4]    z←'<html>'                        ⍝ html docs start with this tag
[5]    z←z,'<head>'                      ⍝ start of the head section
[6]    z←z,'<title>Dynamic</title>'      ⍝ the text for the title bar
[7]    z←z,'</head>'                     ⍝ end of the head section
[8]    z←z,'<body>'                      ⍝ start of the body section
[9]    z←z,'Generated at ',⍕⎕TS          ⍝ the text to display
[10]   z←z,'</body>'                     ⍝ end of the body section
[11]   z←z,'</html>'                     ⍝ end of the html document
    ∇
```

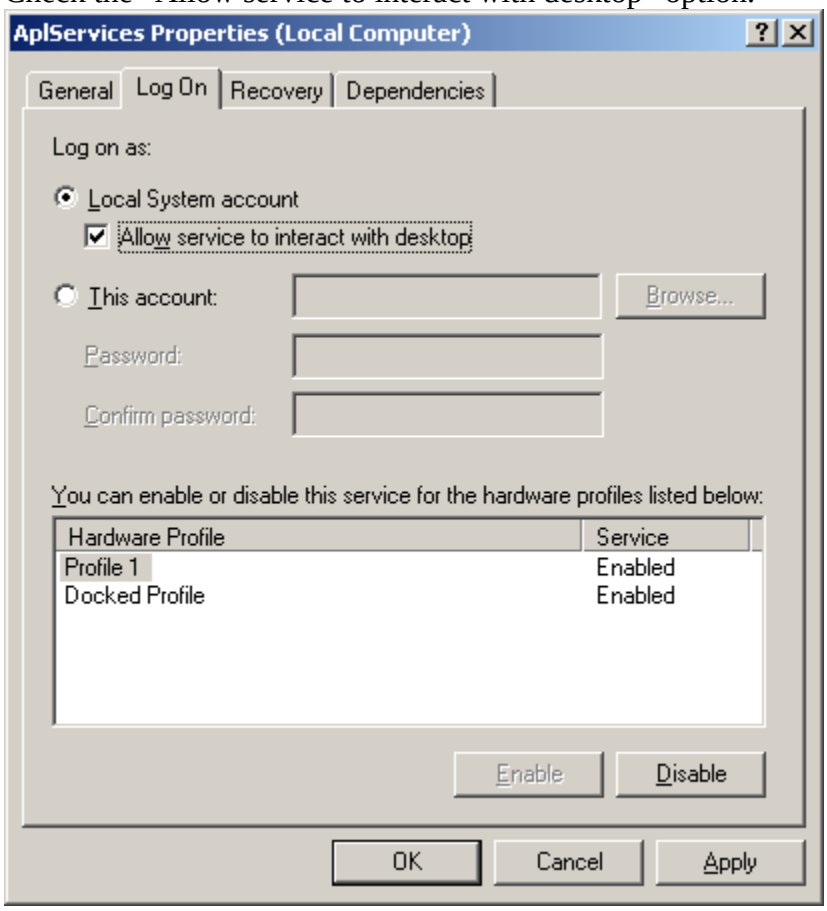
Testing the APL Function

In your web browser, type http://localhost:2000/simple and you should see some simple text with the current date and time. If you refresh the page the time should update.

Looking at the Running Code

Even though we set the *visible* property of basic1 to one, the APL workspace was not shown. The reason for this is that, by default, processes running as services cannot interact with the user. To change this, follow these steps:

1. Open Windows Services.
2. Highlight the AplServices row.
3. Right-click and choose Properties.
4. Go to the Log On tab.
5. Check the "Allow service to interact with desktop" option.



6. Click the OK button to save your settings.
7. Restart the service by right-clicking and selecting Restart.

An APL+Win ActiveX Server session with BASIC.w3 loaded should now be visible.

Modifying the Running Code

Confirm that everything is still working by typing http://localhost:2000/simple into your web browser. Now go to the running APL+Win ActiveX Server session and put a stop on line 10 of the wDynamic function. Refresh your browser and you'll see that it's waiting for a response. Go to the APL session and you'll see that you're stopped at wDynamic[10]. In the session type:

```
z←z,' (about)'
```

and then branch to []LC. Go back to the browser to see that this change has taken affect.

We've now shown that we can modify APL code used by APL Web Services in the same way as we can when running under Windows. If you are modifying code in this way, don't forget to save any changes. If you stop APL Web Services, it will close the open APL+Win COM objects which will lose any unsaved changes in your workspace. In most cases, since)SI is empty, you can simply do a)SAVE at any point.

Don't forget to remove the stop on wDynamic[10].

APL Functions with Arguments

APL functions take only a single right argument and a single left argument. We often get around this limitation by making one or both of the arguments an array of values. The definition of the function may be that it has a single right argument called rarg, but if the first line of the function is something like

```
(num1 num2 name)←rarg
```

then we know that we are converting that single right argument into three separate values. Whatever is calling this function, needs to call it with a three element right argument. In addition, the caller must use the correct type of argument (it shouldn't pass a character matrix when a scalar number is expected).

When APL Web Services calls an APL function that takes arguments, it must be told how to compose those arguments. It also needs to be told what information to use.

Math1

Right-click on *defaultwebsite* and choose New Virtual Directory. Right-click on that virtual directory to rename it to "/Math1". In the same way we did for the /simple virtual directory, set the wsid to "basic1" and the function to "wMath1".

Next right-click on rarg and choose New Value. You can use anything for the name, but set the type to "entity-body" (more about that later).

Now go to http://localhost:2000/math1.htm. This is just a static html page with a form. When you click the "Submit Query" button, the form uses the /Math1 virtual directory. Put in some numbers, choose add or multiply, and click the button. You should get back

a simple page with a single number on it (either the sum or the product of the numbers you typed).

Put a stop on line one of wMath1, go back to the math1.htm form, enter the numbers two, three, and four in the boxes and press the submit button. When we stop on line one, we can see that the right argument is:

```
n1=2&n2=3&n3=4&op=add
```

(Branch to []LC to allow the function to finish.)

If we look at the math1.htm file in a text viewer, we'll see that the form has five fields. The three boxes are called "n1", "n2", and "n3"; the two radio buttons both have the name "op" but have values of "add" and "mult", and the submit button has no name. Also note that the form has a method of "GET" and an action of "/Math1".

The action determines what virtual path to use, and therefore what function to call in the workspace. Each named field gets passed to the server. The text fields use the numbers typed in as the values while the radio buttons pass the value of the checked item.

How Browsers Pass Data to Web Servers

A browser can make a request to a web server using one of two methods: GET or POST. Anything typed in the address bar as well as normal links use the GET method. Forms can be designed to use either method. Which method the form uses is determined by the form designer and the web server administrator.

GET

If the browser uses the GET method, it passes a header and the URI to the server. Any additional data needed by the server must be included in the URI.

Earlier, I said that a URI is of the form http://IPaddress:port/path/file. You can put additional data in the URI by putting a question mark after the file and then putting in "name=value" pairs. Multiple name/value pairs are separated by ampersands. This means your URI may look like http://IPaddress:port/path/file?name1=val1&name2=val2.

This has the advantage that you can simply type it into the address bar, but it means that the user can see the values. Also, there is a limit to the number of characters allowed in a URI.

POST

If the browser uses the POST method, in addition to the header and URI, it also passes additional data to the server which is not visible to the user. In general, this information is from the fields on a web form.

Math1 – revisited

If we run the Math1 example again, we see how the data in the form is put into the URI. If I put the numbers two, three, and four into the text boxes, the URI becomes http://localhost:2000/Math1?n1=2&n2=3&n3=4&op=add. Without going back to the form, simply change the numbers in the address field (for example change the "n3=4" to "n3=44"), and press enter. The page will be updated with the new sum.

Now edit the math1.htm file in a text editor. Change the method from "GET" to "POST" and save the file. Now go back to http://localhost:2000/math1.htm, fill in the form and press the submit button. You still get back a valid number, but the arguments are no longer part of the URI.

The wMath1 function is written such that it will accept data from either a GET or a POST. So even though you just did a post, if you now type in an address of http://localhost:2000/math1?n1=5&n2=10&op=add and press enter, you'll get a response of 15 and the address will show the arguments. That's because typing in an address is the same as a GET. If you wanted your customers to be required to use the form and not be able to just type in the values, you would need to change wMath1 to return an error message if a GET method was used.

The rarg for Math1 is set to *entity-body*. For a GET, the *entity-body* is everything following the question mark in the URI. For a POST, it is the entity-body field of the POST data (hence its name). In either case, it is a text vector of name=value pairs separated by ampersands.

URI Name Matching

If you type in the URI http://localhost:2000/math1.htm, you get our Math1 form. If you type in the address http://localhost:2000/math1, you get back a page that says "Invalid Operation".

The http://localhost:2000 portion of the URI identifies the server and is identical in the two cases. In the first case, "/math1.htm" matches a file in the home directory and so that file is returned. In the second case, "/math1" matches a virtual directory and so calls the wMath1 function in the BASIC.w3 workspace. That function returns "Invalid Operation" if the "op" argument is not either "add" or "mult". Since it was not specified, we get back that error message from the function.

Common Argument Types

There are many argument types available for rarg and larg. I will not cover them all, but will cover some of the common types.

entity-body

This is the argument we used for the /Math1 virtual directory. It is a character vector of "name=value" pairs separated by ampersands.

entity-body-decoded

This is similar to *entity-body*, but it is passed to APL as an Nx2 matrix where the first column contains the names and the second column contains the values.

Math2

Right-click on the "/Math1" virtual directory and choose Copy, then right-click on *defaultwebsite* and choose Paste Virtual Directory. This creates a new virtual directory which is a copy of the original one. Right-click the new virtual directory and rename it to "/Math2". We can leave the wsid alone, but change the function to "wMath2" and change the rarg from "entity-body" to "entity-body-decoded".

Now use your browser to navigate to http://localhost:2000/math2.htm. This is identical to the math1.htm file except that its action is set to "/Math2". Fill in the fields on the form, press submit, and it should appear identical to the previous example.

Put a stop on line one of the wMath2 function and re-run the form. Now look at the right argument to the wMath2 function:

```
      ]display arg
.→--------.
↓.→-..→.  |
||n1||2|  |
|'--''-'  |
|.→-..→.  |
||n2||3|  |
|'--''-'  |
|.→-..→.  |
||n3||4|  |
|'--''-'  |
|.→-..→--.|
||op||add||
|'--''---'|
'∈--------'
```

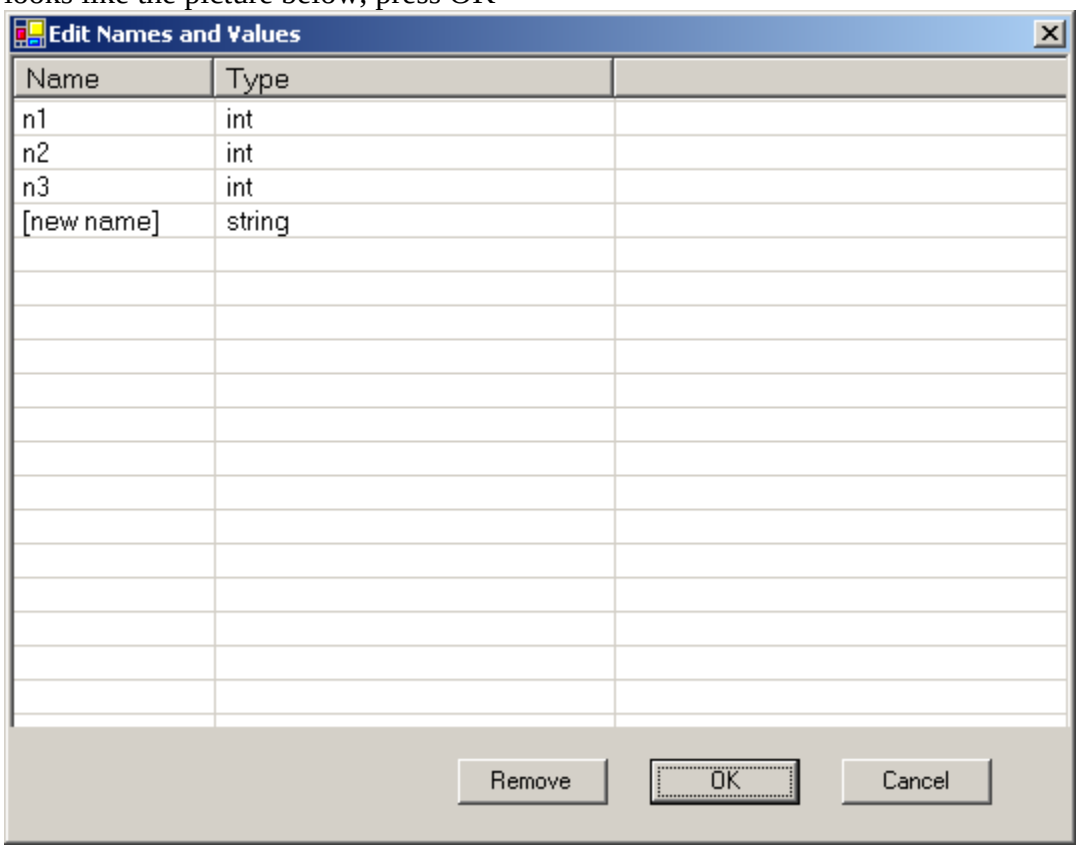
It has the same information as in *entity-body* but it has been converted from a simple text vector into a nested array.

int / float / string

These types are passed just as you would expect. String is a character vector. Float and int are numeric vectors. Unlike the previous two types, the name assigned to these types is important. APL Web Services will look for a name in the request which matches the name specified in the APL Web Services Configuration screen. If it finds it and it is the correct type, it passes that value. If it cannot find a value with that name, or the value is not the correct type (it's "abc" when we specified int), it will return a value of an empty character vector (even if one of the numeric types was specified).

Math3

Create a new virtual directory by copying the "/Math1" virtual directory, pasting the new one, and renaming it to "/Math3". The wsid remains "basic1" but change the function to "wMath3". Right-click on rarg and choose edit values. Highlight the first row and click the remove button. Then double-click "[new name]", put in a name of "n1", and set its type to "int". Repeat the process to add "n2" and "n3" as integers. When the screen looks like the picture below, press OK



Right-click larg, choose New Value and put in a name of "op" and a type of string.

When finished, use your browser to navigate to http://localhost:2000/math3.htm, fill in the form, and press the submit button. Math3.htm is identical to math1.htm other than having its action set to "/Math3".

The result should appear identical to the previous two examples. Put a stop on line one of the wMath3 function, re-run the form, and look at the right and left arguments.

```
      ]display op
.→--.
|add|
'---'
      ]display arg
.→----.
|2 3 4|
'~----'
      →⎕LC
```

There is only one left argument, so it's a simple vector. There are three right arguments, so arg is a three element vector. Each element is an simple integer.

Re-run the math3.htm form, but only fill in the first two boxes, and then look at the arguments.

```
      ]display op
.→--.
|add|
'---'
      ]display arg
.→--------.
|      .⊖.|
| 2  3 | ||
|      '-'|
'∈--------'
      →⎕LC
```

The left argument and the first two elements of the right argument have not changed. The third element of the right argument is an empty vector.

Although the code in the previous example correctly handled empty fields, it was fairly straightforward (the *entity-body* would be "n1=5&n2=6&n3=&op=add"). Now, because we are explicitly asking for the "n3" argument but could not find it, APL Web Services passes an empty character vector as the value.

If you re-run the form again, fill in the first two fields but put "abc" into the third field (so the *entity-body* is "n1=2&n2=3&n3=abc&op=add"), you'll see that arg has not changed. Since we declared n3 as int, and "abc" is not a valid integer, it is the same as omitting it.

Math4

Create a new virtual directory by copying the "/Math3" virtual directory, pasting the new one, and renaming it to "/Math4". The wsid remains "basic1" but change the function to "wMath4". Highlight rarg and then right-click on n2 and choose Delete. Right-click on n3 and choose Delete. You should now be left with a single rarg with name "n1" and type int. The larg should remain unchanged.

Use your browser to navigate to http://localhost:2000/math4.htm, fill in the form, and press the submit button. Math4.htm is identical to math1.htm except that its action is set to "/Math4" and all three input boxes are named "n1".

The result should appear identical to the previous two examples. Put a stop on line one of the wMath4 function, re-run the form, and look at the right and left arguments.

```
      ]display op
.→--.
|add|
'---'
      ]display arg
.→----.
|2 3 4|
'~----'
      →□LC
```

The arguments are identical to the Math3 example. If we re-run this example, but either leave the third box empty or put text instead of a number into that field, the third element of the right argument becomes an empty character vector

```
      ]display arg
.→--------.
|       .⊖.|
| 2   3 | ||
|       '-'|
'∈--------'
```

Setting up a Second Web Server

Right-click on "Web Servers" and choose New Server. Then right-click that web server (called *defaultwebsite1*) and choose Properties.

Web Site

Go to the Web Site tab, change Description to "server2", put in "localhost" for the IP Address, and set the TCP Port to 2002. Since I'm using the same IP address for my two servers (*defaultwebsite* and *server2*), I need to ensure that I don't use the same port.

Home Directory

On the Home Directory tab, set the path to C:\APLconf\WebHome2.

Press OK to save your changes

Starting and Testing the Second Server

To start this server, right-click *server2* and choose Start. Then test this server by using your browser to navigate to http://localhost:2002/titles.htm.

Dynamic Default Pages

In *defaultwebsite* we set a default page so that if the user did not specify a file name in the URI, they would get the default.htm file, if it exists, in the requested directory. For *server2* we did not set a default page. Instead we'll create a virtual directory for the case where no file is specified.

Add a virtual directory by right-clicking *server2* and choosing New Virtual Directory. Rename it from "/default" to "/" (a single slash). Set wsid to "basic1" and set the function to "wWSinfo". This function takes no arguments so no other parameters need to change.

If you now navigate to http://localhost:2002 (no page specified), you'll get information returned from APL.

Virtual Directory Names

All of the virtual directories created so far look like a path which is one level below the root web path. We can also have them look like files or look as though they are deeper in the directory tree.

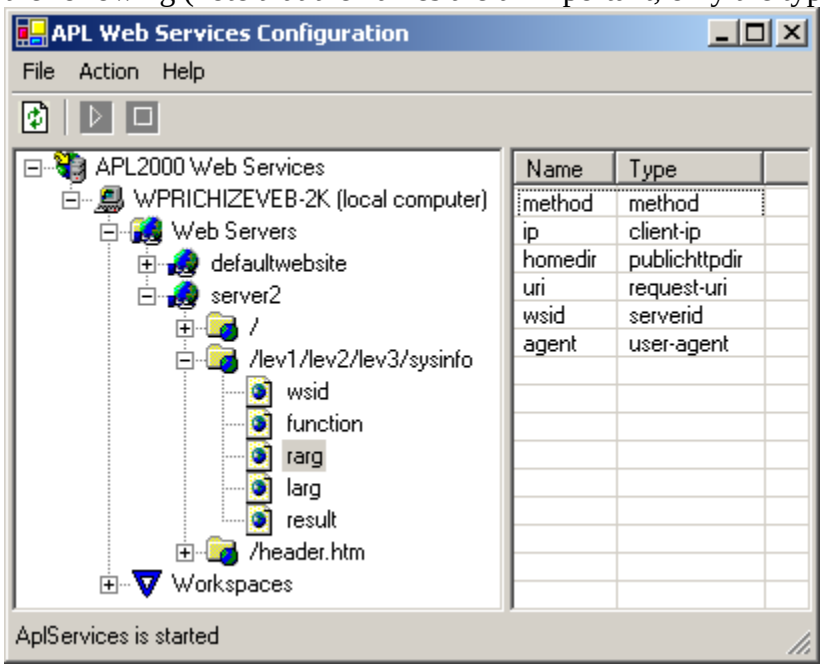
Virtual Directories That Look Like Files

Under *server2*, create a new virtual directory and rename it to "/header.htm". Set the wsid to "basic1" and the function to "wHeader". Create a rarg with any name and with the type of "header".

Now navigate to http://localhost:2002/header.htm, which to the end user will look like a file name, and see the result returned from APL.

Virtual Directories Deeper Than One Level

Under *server2*, create a new virtual directory and rename it to "/lev1/lev2/lev3/sysinfo". Set the wsid to "basic1" and the function to "wWebInfo". Create your rarg to look like the following (note that the names are unimportant, only the types matter):



Now navigate to http://localhost:2002/lev1/lev2/lev3/sysinfo to see the result returned from APL.

APL Web Services does not care what you call the virtual directory. It simply does an exact match of the *request-uri* (up to the first question mark), to the virtual directories assigned to that server. If it finds a match, it uses that virtual directory.

In this URI:

http://IPaddress:port**/path1/path2/…/pathN/page**?data

the underlined, bold portion is checked against the virtual directories you have created. You can see, therefore, that all virtual directories must start with a slash and cannot contain a question mark.

Additional Argument Types

These argument types all convey information about either the browser, the server, or the request. They have been shown in the http://localhost:2002/lev1/lev2/lev3/sysinfo or http://localhost:2002/header.htm examples. All these types, and others, are also documented in the help files distributed with APL Web Services.

client-ip

This is the IP address of the client browser. In our examples, it was displayed as 127.0.0.1 (localhost). Be aware that if the user is coming through a proxy, this may be the IP address of the proxy.

header and header-parsed

This is the entire HTTP header as either a flat text string (as shown in the header.htm example) or an Nx2 matrix of name/value pairs.

method

This is the method used to send the URI. In most of the examples in this paper, it was "GET". In some of the examples using forms, we also used "POST". Other methods such as "HEAD" are also defined.

publichttpdir

This is the path to the home directory as defined in the property sheet for the web server.

request-uri

This is the URI with the IP address and port removed. In includes any additional parameters (the text after any question mark).

serverid

This is the name of the web server as defined in APL Web Services Configuration.

user-agent

This is the type of browser being used. You would use this field if you wish to customize your content based on the browser being used.

Common Result Types

So far, all of our virtual directories have returned a result of type "document" but other types are available.

document

When using this result type, you should return a character vector. This can either be a valid HTML document, or simply plain text which will be displayed without decoration in the browser screen.

document-filename

The *document-filename* result type allows you to specify a filename which APL Web Services will return. This allows you to dynamically decide on the file to return, return files which are not in subfolders of your home directory, or return files of other content types (such as Excel, PowerPoint, or Word files).

Returning a File

Add a new virtual directory to *server2*, and rename it to "/yellow". Set the wsid to "basic1", the function to "wYellowFile", and the result type to "document-filename". Now navigate to http://localhost:2002/yellow. Even though this server's home directory is C:\APLConf\WebHome2, you will see the C:\APLConf\WebHome\Yellow\default.htm page. If you look at the wYellowFile function, it simply returns the name of this file.

content type

The *content-type* is returned with *document-filename* in order to tell the browser what type of document is being returned. APL Web Services will determine the type automatically based on the filename extension for some types (such as .GIF, .JPG, .XLS, etc.). For file types not handled automatically, or if you wish to override the default type, you can return the content type as a text string.

You can also set the content type based on extension globally for each server on the Documents tab of the server properties dialog.

document-filename-delete

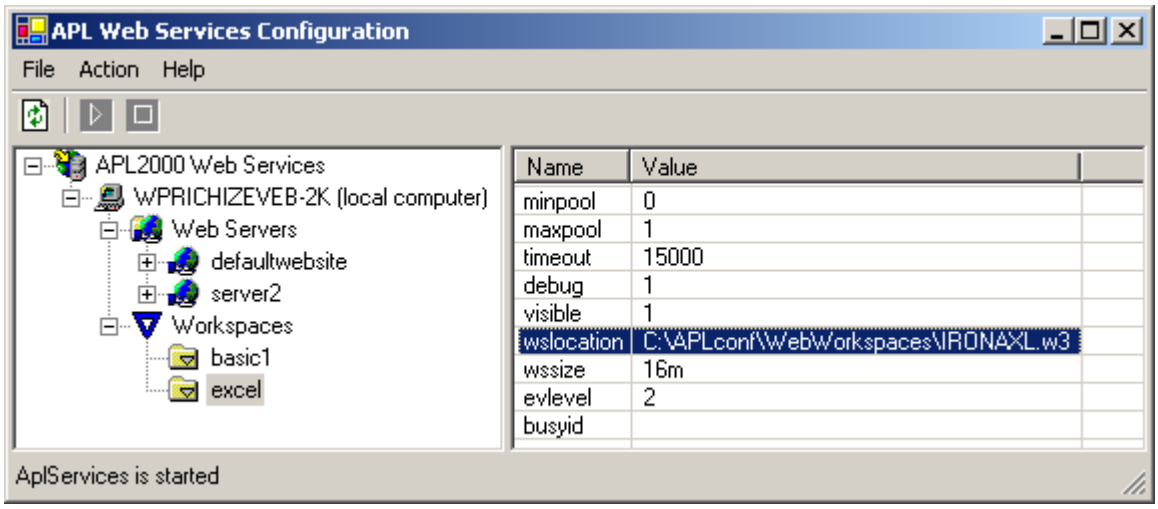
This is a flag returned with *document-filename* to indicate whether APL Web Services should delete the file after returning it. The file will be deleted if this is set to one, but will not be deleted if it is set to zero.

Adding a Second Workspace

So far, both of our web servers are using functions in the same APL workspace. You do not need to put all of your web related functions into a single workspace. You can add additional workspaces, accessible by any web server, so that your APL functions are partitioned between workspaces in whatever manner you like. In fact, this is one of the benefits of APL Web Services. Other than adding some HTML output routines, many of your functions should be usable from the web with little or no change.

Right-click on Workspaces and choose New Workspace. Right click the new workspace and rename it to "excel". Then set *minpool* to "0", *debug* to "1", and *visible* to "1".

Set *wslocation* to the C:\APLconf\WebWorkspaces\IronAXL.w3 file. This is the "Ironing Out the Wrinkles in APL+Excel" from the 2003 user conference with the wXLShow from function added to it. You could instead use XLAPL01.w3 ("Secrets of the APL+Excel Interface" from the 2001 user conference). The workspace can be in any directory, but I have put it into the same directory as BASIC.w3.



Once you have added the workspace and set its properties, right-click on it and choose Start. The *excel* workspace is now available for use from your web servers.

Right-click on *server2* and choose New Virtual Directory. Then rename it to "/xl". Set the wsid to "excel", the function to "wXLShow", the rarg to a type of "header-parsed", and larg to a type of "entity-body-decoded". The result will have two values, the first of type "document-filename" and the second of type "document-filename-delete".

Now navigate to http://localhost:2002/xl?name1=value1.

Three things should happen:

1. A webtemp.xls file is created in the temporary directory.
2. A two sheet Excel workbook appears in the browser.
3. An APL+Win COM object with the excel workspace loaded appears.

Modify the wXLShow function in the excel workspace, so that the second element of the result is a one (instead of a zero). This will cause APL Web Services to delete the file after returning it to the browser. Now navigate to http://localhost:2002/xl. You will get a new Excel workbook in your browser, but the webtemp.xls file is no longer in the temporary directory.

Using Your Existing Code

The wXLShow function in the excel workspace calls seven standard functions in that workspace, and then returns the name of the Excel file that was just created and a flag of whether APL Web Services should delete the file after sending it to the browser. You will often find that you can migrate your existing workspaces to APL Web Services by adding some cover functions, removing the user interface, and doing some simple HTML formatting.

Load Balancing

An instance of APL can only do one thing at a time. If you have a virtual directory which calls a long-running APL function, then any other requests to that workspace will have to wait until the first request is done. To help with this problem, you can have Workspaces in APL Web Services that load multiple APL+Win COM objects. This way, if the first one is busy, a second (or third, or fourth, …) can be opened to process the request.

The *maxpool* setting is the maximum number of APL+Win COM objects that will be used by that workspace. We set this to one for both *basic1* and *excel*, but you can set it as high as you need, up to whatever limit your machine can handle. (A machine with 64MB of memory cannot run as many processes as one with 1GB.) APL Web Services will open new APL+Win COM objects, up to the *maxpool* setting, as they are needed. If it is unused for a certain amount of time, they will be closed.

The *minpool* setting is the minimum number of APL+Win COM objects that will be opened by that workspace. This was set to one for *basic1*, but was set to zero for *excel*. If you know that one of your workspaces will usually need to run multiple copies, then it is more efficient to set this number higher than one so that APL Web Services won't close them if they happen to be unused. This saves the time of re-opening them.

The *basic1* workspace is always visible, because its *minpool* is one. Since the *excel* workspace had a *minpool* of zero, it was not opened until it was needed. In addition, if it is not used for a while, it will be closed down.

During development, it is often best to set both of these values to one. If multiple copies are open, changing code in one copy of the workspace will require you to re-load it in the other copies. Also, if *minpool* is zero and you make a change, APL Web Services may close that copy before you save your changes.

Cookies

A cookie is a small piece of information which a web server can store on the browser's machine. Most web browsers allow you to set whether you want to accept cookies all the time, reject them all the time, or be asked each time a cookie is saved. Cookies are generally used to store information which the web server wants saved from one session to another or, within a session, between pages of the web site.

Cookies do have a number of limitations:

- Cookies can only be seen by the domain which saved them.
- Each domain can save at most 20 cookies.
- Each domain is limited to 4kb total for all their cookies.

Because of the last two limits, if you have a large amount of information you want saved for a user, you would generally store it on the web server and have the cookie be a unique ID to get that information.

Each cookie must have a name and a value. In addition cookies can have optional parameters of expires, path, domain, and secure.

expires

The date and time at which the cookie will be removed from the client machine. If this is not set, the cookie is called a "session" cookie and will expire (be removed) when the browser is closed. The date must be expressed in Greenwich Mean Time as

DAY, DD-MMM-YYYY HH:MM:SS GMT

where DAY is the day of the week (Sun, Mon, Tue, Wed, Thu, Fri, Sat), DD is the day in the month (01, 02, 03, …, 31), MMM is the three-letter abbreviation for the month (Jan, Feb, Mar, …), YYYY is the year, HH is the hour in military (24-hour) time, MM is the minute value, and SS is the second value. For example:

Sun, 07-Nov-2004 13:05:00 GMT

To delete a cookie, set its expiration in the past such as:

expires=Thu, 01-Jan-1970 00:00:00 GMT

path

Setting the path parameter specifies that the cookie can be returned only for paths that are at the path level specified or below. If path is not set, then it defaults to the URL path of the document creating the cookie.

domain

The domain parameter sets the domain or machine for which the cookie is valid.

If this is not set and support.microsoft.com sets the cookie, then only support.microsoft.com can see the cookie. If this was set to

domain=microsoft.com

then support.microsoft.com, home.microsoft.com, www.microsoft.com, etc. would all be able to see the cookie.

For security reasons, the domain name must have at least two or three periods and only hosts within the specified domain can set a cookie for a domain.

secure

The secure parameter is a flag indicating that a cookie should only be used under a secure server condition, such as SSL. This parameter defaults to FALSE.

ShowCookies

Add a new web server by right-clicking on "Web Servers" and choosing New Server. Open its properties to set its name to "CookieDemo", the IP address to "localhost" and the port to 2003. On the Home Directory tab, set the path to C:\APLconf\WebHome2 (the same as we used for server2). Click OK and then right-click CookieDemo and choose Start.

Create a new virtual directory in the CookieDemo web service called "/ShowCookies". Set its wsid to "basic1", its function to "wCookieShow", and a rarg with any name of type "cookie".

Navigate to http://localhost:2003/ShowCookies and you should see all the cookies in the "localhost" domain (which is probably empty). The wCookieShow function returns the unformatted cookie information (a simple text vector of name=value pairs separated by semi-colons) as well as a formatted version where each name/value pair is on its own line.

AddCookie

Create another virtual directory called "/AddCookie" with a wsid of "basic1", a function of "wCookieAdd", a rarg of type "string" with the name "cook_name" and a larg of type "string" with the name "cook_value". Set the result to have two items: the first is a "document" type and the second is a "cookie" type (the names do not matter).

Navigate to http://localhost:2003/AddCookie.htm (by either typing it in or clicking the link on the Show Cookies page). This is a static page in the C:\APLConf\WebHome2 directory with two input fields and a button. If you fill in the name and value fields and press the "Add Cookie" button, it will set the value of that cookie. If the cookie already exists, it will reset the value. You can use the Show Cookies link to see that it has been added.

DeleteCookie

Create another virtual directory called "/DeleteCookie" by copying the "/AddCookie" virtual directory and renaming it. Remove the larg, but keep everything else the same. Navigate to http://localhost:2003/DelCookie.htm. This is a static page with a single input field and a button. Type in a name of a cookie you have added and press the "Delete Cookie" button.

This calls the same function as AddCookie, but does not pass a left argument. The wCookieAdd function detects that no left argument (cookie value) was passed and so deletes the cookie. The cookie is deleted by setting its expiration in the past.

Uploading Files

When you upload files or other large data, instead of passing it to an APL function, APL Web Services will put the data into a temporary file and pass the name of that file to the APL function.

Upload1

Add a new virtual directory in *defaultwebsite* called "/upload1". Set its wsid to "basic1", its function to "wUpload1", and a single rarg with a type of "entity-body-filename". Give it two largs of types "header-parsed" and "method" (the names for all of these are unimportant). Leave the result as a single item of type "document".

Navigate to http://localhost:2000/upload1.htm, fill in the form and press the "Do It" button. The response shows the header information, including that the content-type is "multipart/form-data". The result also shows that the uploaded information was written to a temporary file. If you look at that file in Notepad, you'll see that it has four sections. The first three correspond to the contents of the three edit boxes and the fourth to the button.

Now navigate to http://localhost:2000/upload2.htm, fill in the form and press the "Do It Again" button. If you look at that temporary file, you'll see that it only has two sections. This is because the file will only contain information for those form elements which have names. In upload1.htm, all four controls were named. In upload2.htm, the bottom edit box and the button, were not given names and so they were not written to the temporary file.

The other thing to note in these two examples is that the content-type in the header also lists a value for boundary. That value is the separator in the temporary file between the fields of information. The browser finds some string value which is not in the data being uploaded, uses that as a separator and then puts it in the header. This will allow us to parse the temporary file into its components.

Now navigate to http://localhost:2000/upload3.htm, fill in the form and press the button. This page is the same as the upload1.htm page, except that the middle edit box has been replaced by a file box. (I suggest using a small text file for this example.) When you look at the temporary file, you should note that the middle section contains the name of the uploaded file as well as its contents.

Upload2

Rather than just displaying the header and informing you of the location of the temporary file, we want to parse the information in that file. You can either remove the comments on lines 10 and 11 of wUpload1 (which will simply call wUpload2 with the same arguments and return its result, or you can modify the "/upload1" virtual path so that its function is "wUpload2".

Again navigate to http://localhost:2000/upload3.htm, fill in the form and press the button. The response page will tell you the values in all the edit boxes as well as the location of the file created from the uploaded file. (The function wUpload2 puts all uploaded files into the C:\APLConf\UploadFiles directory.)

If you look at the wUpload2 function, you'll see that it does the following steps:

- Read the contents of the temporary file into a variable and erase the temporary file
- Read the header to get the boundary string
- Partition the data at the boundary string into a nested vector
- Process each piece of the data by doing the following:
    - Split the piece into its header and data
    - Get the name from the header
    - Get the value from the data portion
    - From the header, determine if it's a file and if so
        - Get the original file name
        - Get the content-type of the file
        - Create the file in the UploadFiles directory
- Return the information about that piece of the form's data

You can now navigate to http://localhost:2000/upload4.htm which will allow you to upload multiple files with descriptions.

Limitations in the examples of uploading files

Although these examples show how to upload files or other large pieces of data, the APL function wUpload2 was simplified for readability and is not suitable for a production environment. None of the functions in the sample workspace should be used in production without modification (error handling, edge conditions, etc.). There are additional specific changes which should be made to this function.

- This function reads in the entire temporary file. Since uploaded files may be huge (many megabytes or gigabytes), you would want to read the file in manageable chunks to process it.
- This function writes out all uploaded files into a single directory. Since different people may upload different files with the same name, or a single person may upload files from different directories with the same file name, you would want to segregate files by person uploading them and then by directory (or some other mechanism).

Topics Not Covered In This Paper

There are a great many topics that are beyond the scope of this paper, but you should be aware of some of them so that you can get more information on them before you put your web application into production.

Stateless vs. Statefull Connections

Most web sessions are stateless. That means that each request creates a connection, transfers information, and then breaks the connection. This is very useful for web servers because the amount of time satisfying a request is much less than the time spent by the user looking at the result. If I ask for a static page, the web server delivers it in under a second, the transmission time over the network may take 10 seconds, and then I may spend a minute reading it before making another request. In a stateless connection, the web server is only tied up for the time to deliver the page (less than a second). In a statefull connection, the server is busy until the connection is broken, so it cannot satisfy other requests during the time that I'm reading the result. You can handle many more users in a stateless environment.

Unfortunately, there are drawbacks to a stateless connection. When programming for Windows, we often keep information such as the user name, what security rights they have, and any other global information, which you cannot keep with a stateless connection. You need to ensure that your application does not keep global variables about the user. If this is not possible, and you have a small number of users, you can create a statefull connection by using the *one-to-one* result type.

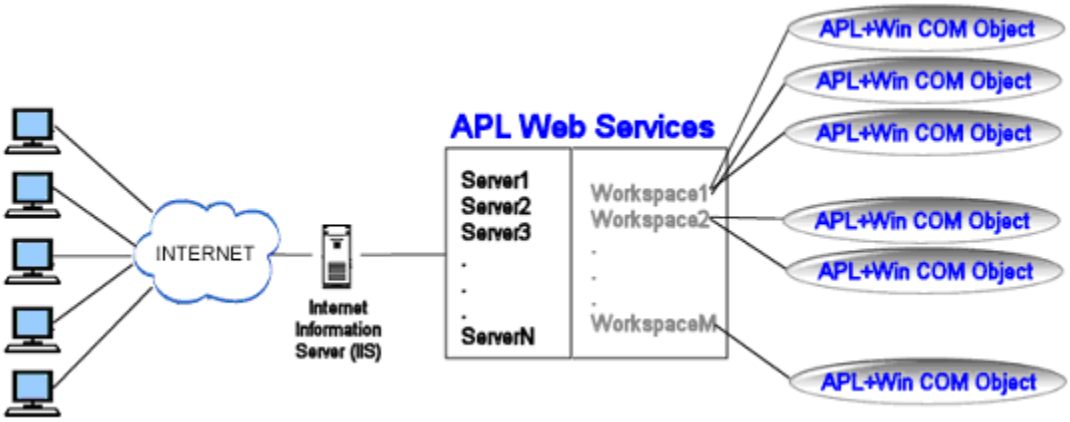
Busy Workspaces

If a workspace already has as many APL+Win COM objects as set in its *maxpool* property, and another request is made, the server will wait for a little while but will eventually timeout. You can set the *busyid* property on a workspace to a virtual path to use when all the APL+Win COM objects are unavailable. This virtual path may be as simple (a static page that says "busy, try again later") or as complex as you wish.

Using APL Web Services with IIS

In this paper, we have used APL Web Services as our web server. If you are installing at a customer site, they may already be using another web server such as Microsoft Internet Information Server (IIS).

In the simplest scenario, you may just modify the pages served by IIS so that they contain links and form actions which point to APL Web Services. However, some IT departments may require that all web access go through IIS. In that case, you can make APL Web Services an ISAPI (Internet Server API) extension to IIS. This is a means by which IIS can call your application in APL Web Services but all the information still goes through IIS allowing the customer to use their existing IIS experience and tools (to monitor, log, filter, etc the information being passes.)



SSL and Security

If you are running APL Web Services as an ISAPI service behind IIS, then IIS handles all SSL issues. If you are running as your own web server, secure sockets are only supported in the advanced version of APL Web Services.