APLNext Application Server (AAS)

Contents

Summary

APLNext Application Server (AAS) is the web server for APL+Win-based application systems. Using AAS it is easy to publish information or APL+Win programs which implement a business rule or algorithm to an enterprise intranet or the public Internet.

AAS runs as a Windows service. A Windows service is a Windows program which runs in the background of a machine using the Microsoft Windows operating system to perform specified tasks. A Windows service runs under its own credentials for enhanced security. The specified tasks are known as web services. The specified tasks are initiated by client requests and result in responses to those client requests such as a document, a calculation, a database update, etc. Clients can be humans using a web browser or other GUI or program-controlled clients based on another workstation or server each of which has been configured to connect to AAS.

The transmission of client requests and web service responses uses the TCP/IP and HTTP or HTTPS protocols. Information published using AAS can be formatted in numerous ways, the most common being HTML web pages. The web server can also receive and transmit other information structures, e.g. entire arrays of values.

Html-format documents can be pre-defined, static web pages, or can be assembled under program control by the programs supporting a web service. Numerous tutorials are available which illustrate html programming. Using APL+Win functions to construct html-format documents are easy to develop since such documents are pure text with xml-format delineations.

AAS can support multiple web services on the same Windows machine as well as support multiple users of the same web service on that machine since AAS is a multi-threaded .Net program. To support this multi-web service and multi-user environment, AAS can be programmer-configured to instantiate multiple instances of APL+Win ActiveX engine each of which loads a programmer-created workspace containing programmer-created APL+Win functions to support the business rules or algorithms of the web service application system.

AAS receives client requests and can direct those requests, including client input, to a programmer-developed APL+Win function that is associated with a specific web service. After that APL+Win function completes processing AAS directs the response back to the client. AAS can be programmer-configured to automatically handle all the conversions between the APL+Win data formats and the web-standards formats used by the client.

More information about the AAS is available on the APL2000 Web Services forum:

- What is AAS?
- AAS overview
- Multi-threading, scaling, queueing, load balancing and state
- Benefits, Licensing, Pricing

Installing AAS

The APL2000 Web Services forum contains detailed documentation for the installation process. Here is a summary of that process:

- Install the Microsoft .Net 4.5.2 Framework [full version].
- Download the installer software and follow the installer software instructions observing the Windows Vista/Windows 7 instructions.
- If necessary activate this installation of AAS with the license key and password provided by APL2000.
- Install the appropriate developer or runtime version of APL+Win software to the machine.
- Register APL+Win as an ActiveX engine on the machine.

AAS Web Service Security

AAS is designed so that clients only have access to server-side functionality, directories and files which have been specified by the programmer. For example the programmer may include html-format documents that will be available via an AAS web service in the home directory of the web service, but other directories on that machine are not, by default, accessible by the web service clients.

AAS clients have access to the server only via web services which have been programmer-configured, which means that APL+Win functions and their containing workspaces are secure, i.e. not directly accessible or downloadable via AAS. Client access to the functionality of an AAS-supported web service can be precisely controlled by the programmer.

The properties which are programmer-maintained for an AAS server are called the configuration. For security reasons the current configuration file of an AAS server is encrypted so that it cannot be modified except by an authorized person using one of the AAS administration tools. Using these tools an un-encrypted version of an AAS configuration can be exported.

Administration – Included Tools

Although an AAS web server is designed to operate without the continuous presence of a server-side human operator, a human server administrator must establish the desired configuration properties of the AAS web server so that it will properly satisfy client requests.

AAS includes the APLNext Application Server Administration tool which is a programmer GUI for configuring the web server and web services and starting/stopping the web server or a web service. This tool is automatically installed to the target server machine when the AAS installer is run.

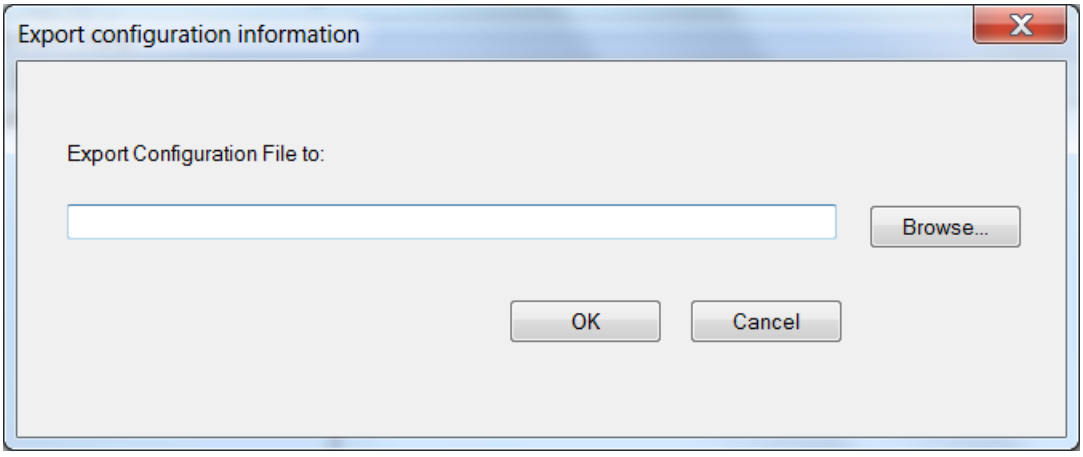
Most of the AAS administration tasks may also be performed under program control using the APLNext.AppSvrController.

Administration – Web Server Actions

Some actions available in the APLNext Application Server Administration tool apply to all configured web services of a specific AAS web server installation:

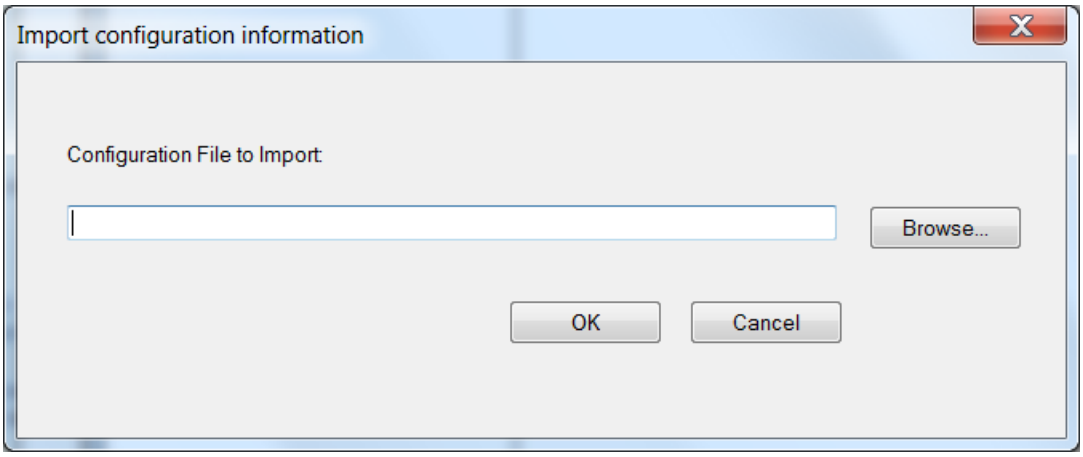
Action – Export

Use this option to export the entire AAS configuration to an xml-format text file. This option is useful to back-up a configuration or use a configuration to create an additional AAS server.



Action – Import

Use this option to import a previously-exported AAS configuration. This option is useful when creating a new production AAS server based on an updated development environment configuration.



Action - Flush Log

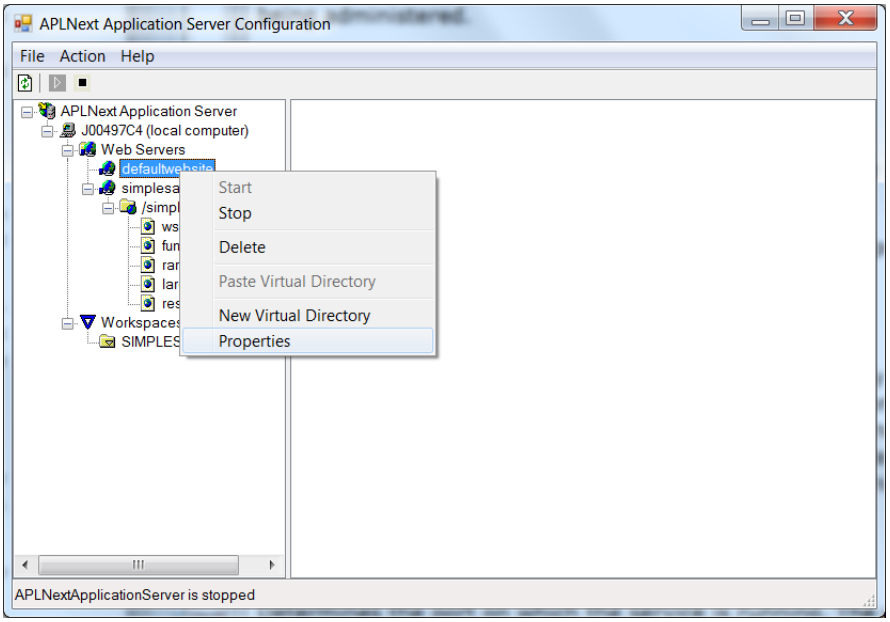
Use this option to move any remaining memory-based log entries to the log file. To avoid degrading AAS performance log entries are accumulated in memory in 65Kb blocks and then written to the log file as necessary rather than continuously writing individual log file entries to the file.

Action - Stop/Start

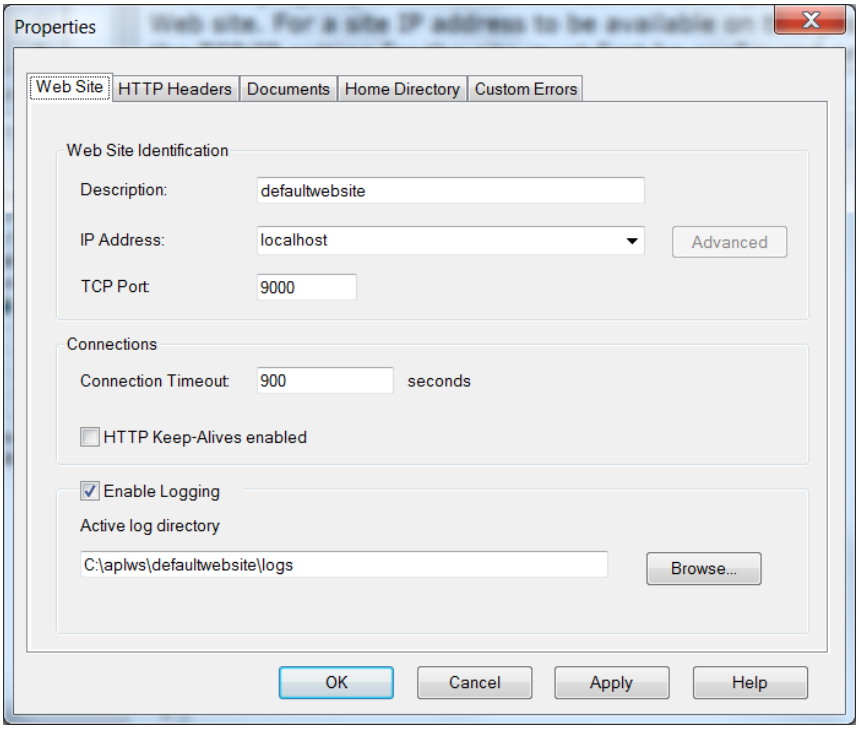
These actions apply to the selected node in the web server tree on the left of the main form of the administration tool. If the 'Web Servers' node is select these actions will affect all web services configured in AAS. If a specific web server is selected, only that web service is affected. These options facilitate minimized interruption of web service availability for clients.

Administration – Web Site Properties

The APLNext Application Server Administration tool property sheets for a specific web service configured in AAS are accessed by selecting a web site in the tree on the left panel of the main form of the tool and using the context menu [right click] to select the 'Properties' option:



Web Site Property Sheet



Description

Any desired text which will be displayed in the left side tree on the main form of the APLNext Application Server Administration tool

IP Address

The valid IP Address or domain name of the machine on which AAS has been installed.

TCP Port

A positive integer indicating the port number. Client requests which are received by AAS at that port number will be directed by AAS to this web site. By convention certain port numbers have been assigned to specific web service tasks. Note that if another web server, e.g. Microsoft Internet Information Services (IIS) is also installed and running of the machine, be sure to avoid port number conflicts with that other web server. In some cases the selected port will have to be opened in the security configuration of the machine on which AAS has been installed.

Connection Timeout

Enter the number of seconds before the server disconnects an 'inactive' client to insure that all such 'inactive' connections are closed if the http protocol actions fail to close a connection. Timeouts are essential to maintain server responsiveness in the case of heavy user load.

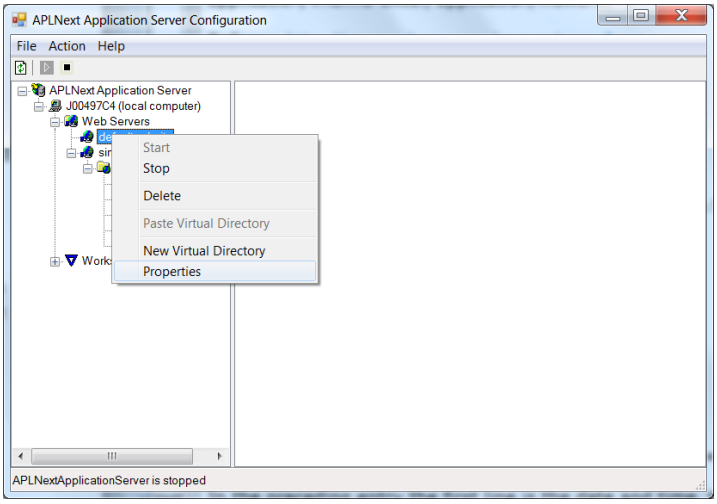
HTTP Keep-Alives enabled

Check this item if a client will be allowed to maintain an open connect with the web server, rather than re-opening the client connection with each new client request.

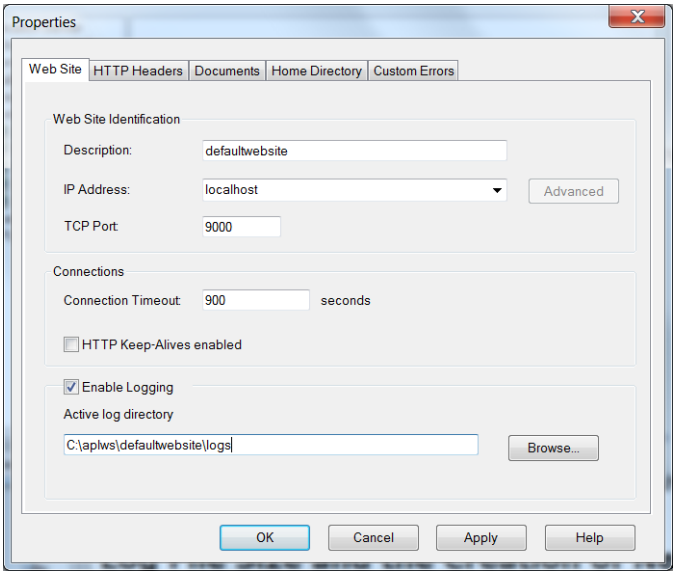
Enable Logging and Active Log Directory

Logging of web server activity, if enabled, occurs in a separate thread from the processing carried out by the instances of the APL+Win ActiveX engine created by AAS. Log entries are created as a client accesses the web services configured in AAS, however these log entries are maintained in memory until they are flushed to the log file.

To enable logging of web server activity in the APLNext Application Server Admin tool, select the web server and use its context menu [right click] to select the 'Properties' option:



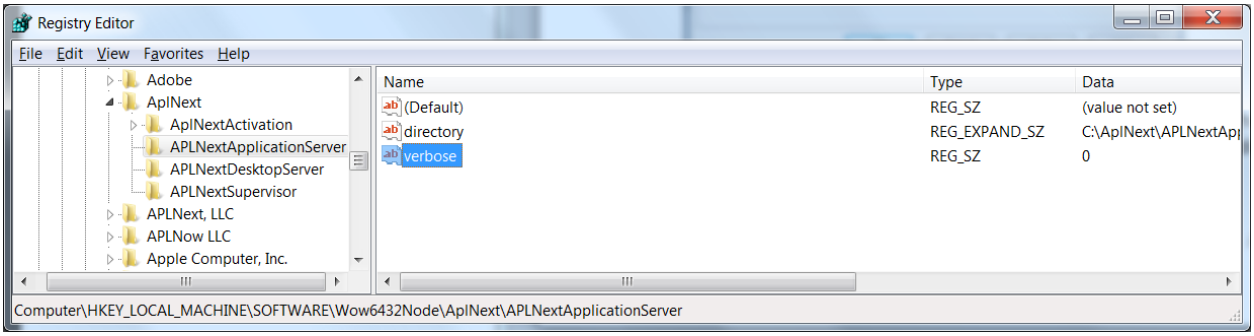
Select the 'Web Site' tab and check the 'Enable Logging' check box and enter the name of the 'Active log directory' on the target server':



The log file entries can programmer-selected as 'verbose' in which case additional log entries are created by the software. This setting is maintained in the Windows registry of the server containing the AAS software in the

HKEY_LOCAL_MACHINE\Software\Wow6432Node\AplNext\APLNextApplicationServer

key value of 0 for non-verbose, the default value on software installation, or programmer-set to 1 for verbose log file entries.

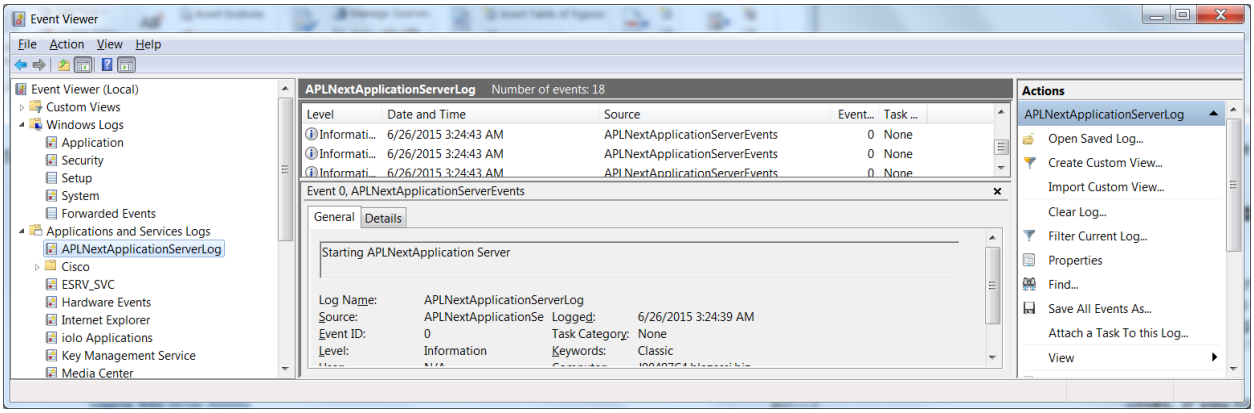


The AAS log file content format is ASCII and includes basic items such as the user's IP address, user name, request date and time, HTTP status code, and number of bytes received. In addition, it includes detailed items such as the elapsed time, the number of bytes sent, the action (for example, a download carried out by a GET command) and the target file. The time is recorded as local time.

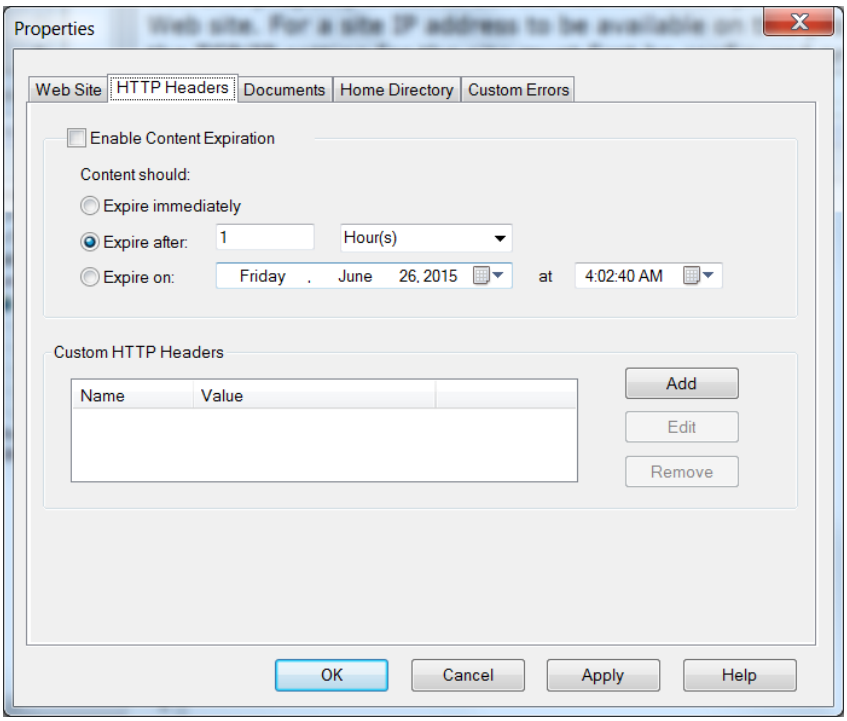
When you open the AWS log file in a text editor, the entries are similar to the following example:

```
2003-05-18 19:58:33Z
POST /samps/sampform HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-excel, application/msword, */*
Referer: http …
Accept-Language: en-us
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; .NET CLR 1.0.3705)
Host: rr.openhere.com
Content-Length: 43
Connection: Keep-Alive
Cache-Control: no-cache
Cookie: ziplocal=08003
```

Besides the logging of web server activity, AAS may log certain events, such as starting/stopping the server or error conditions related to AAS in the Microsoft Windows Event log. These Microsoft Windows events are created as necessary by AAS and are not programmer-controlled. The Microsoft Windows Event Viewer is accessible via Start > Control Panel > Administrative Tools > Event Viewer:



HTTP Headers Property Sheet



Enable Content Expiration

Check this option to enable content expiration.

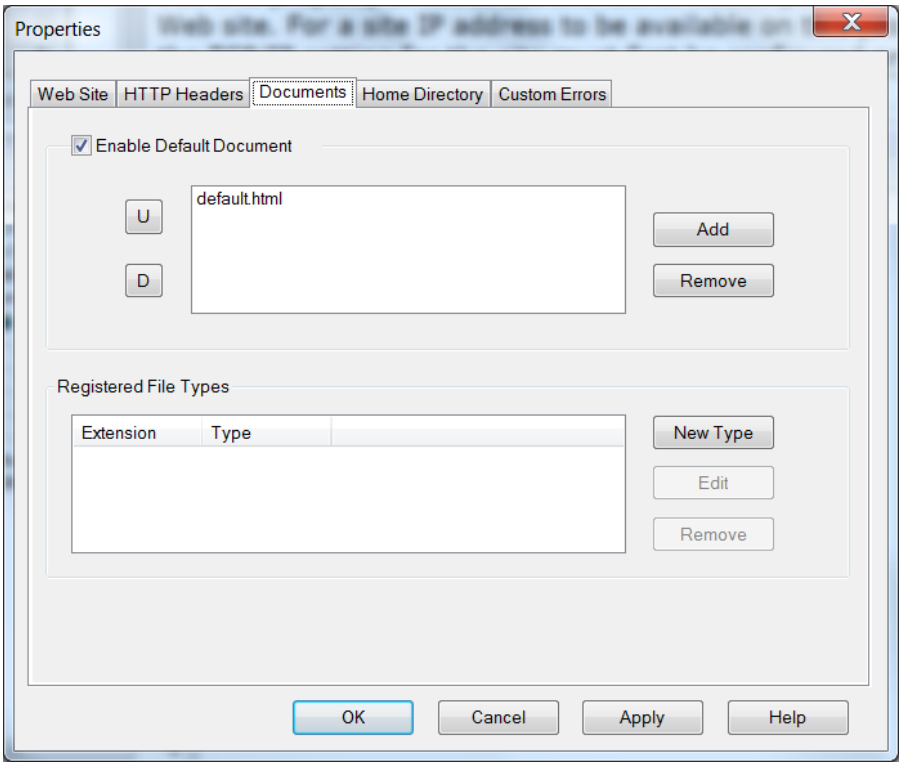
Content Should

Select the appropriate content expiration option and its associated parameters to assure that obsolete content will not be used by clients. After content expires, the client must re-request the desired content so that only up-to-date content is returned. This feature is useful when content includes frequently updated information. For a browser-based client GUI, the browser will compare the current date with the expiration date of the content to decide if content cached on the user machine can still be used or if that content should be re-requested from the web service.

Custom HTTP Headers

Use this property to send a custom HTTP header from the web service to the client browser. For example, use a custom HTTP header to allow the client browser to cache the page but prevent proxy servers from caching the page. Custom headers are described by a name and a value.

Documents Property Sheet

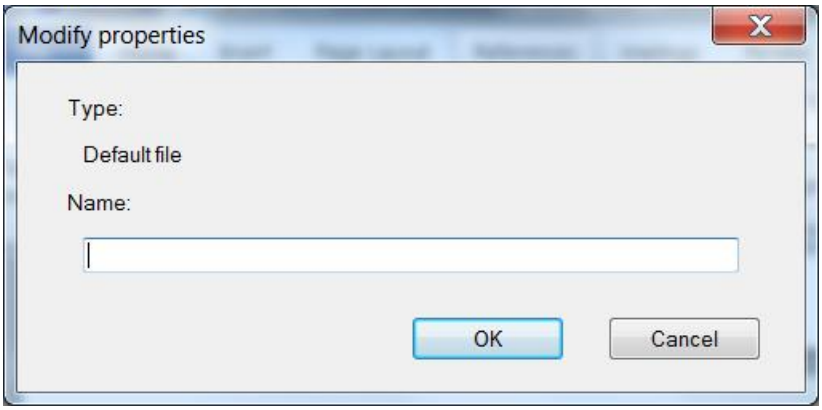


Enable Default Document

Select this item if a default document is defined for this web service.

Default Document List

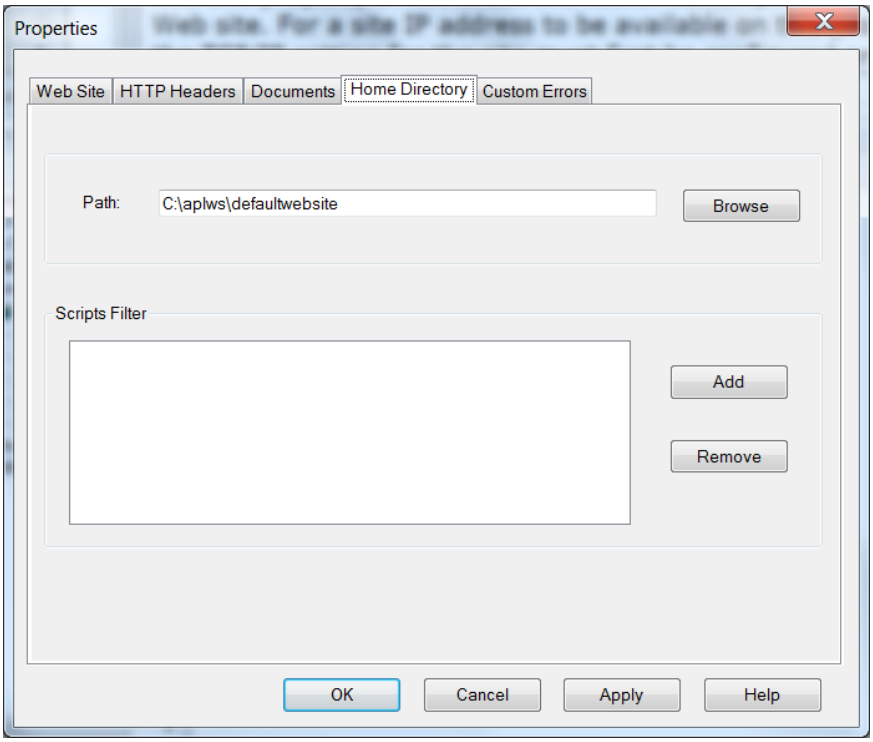
Add/Delete default documents using the subordinate dialogue:



Registered File Types List

Use this option to add, edit or delete Internet file extensions which will be recognized by this AAS web server.

Home Directory Property Sheet

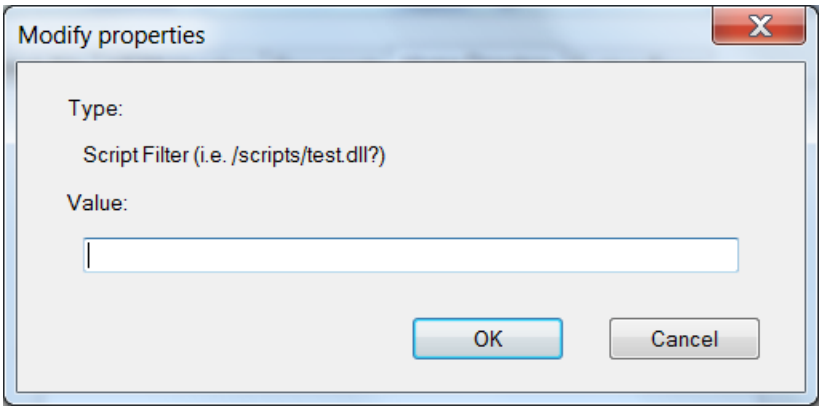


Home Directory Path

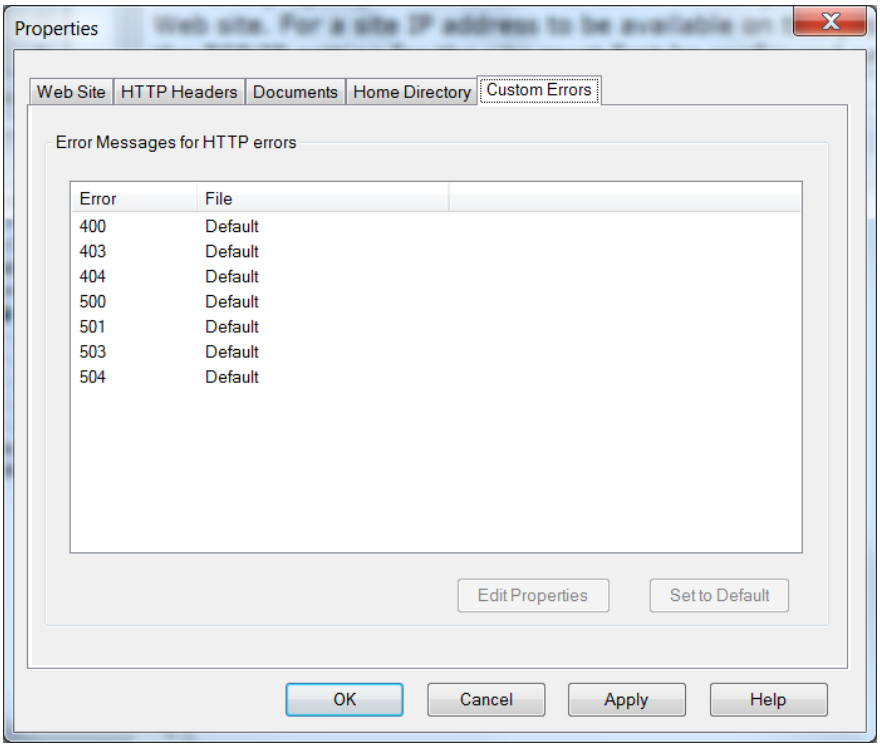
Enter the path of the home directory for this web server. This folder will be accessible by clients of AAS.

Scripts Filter List

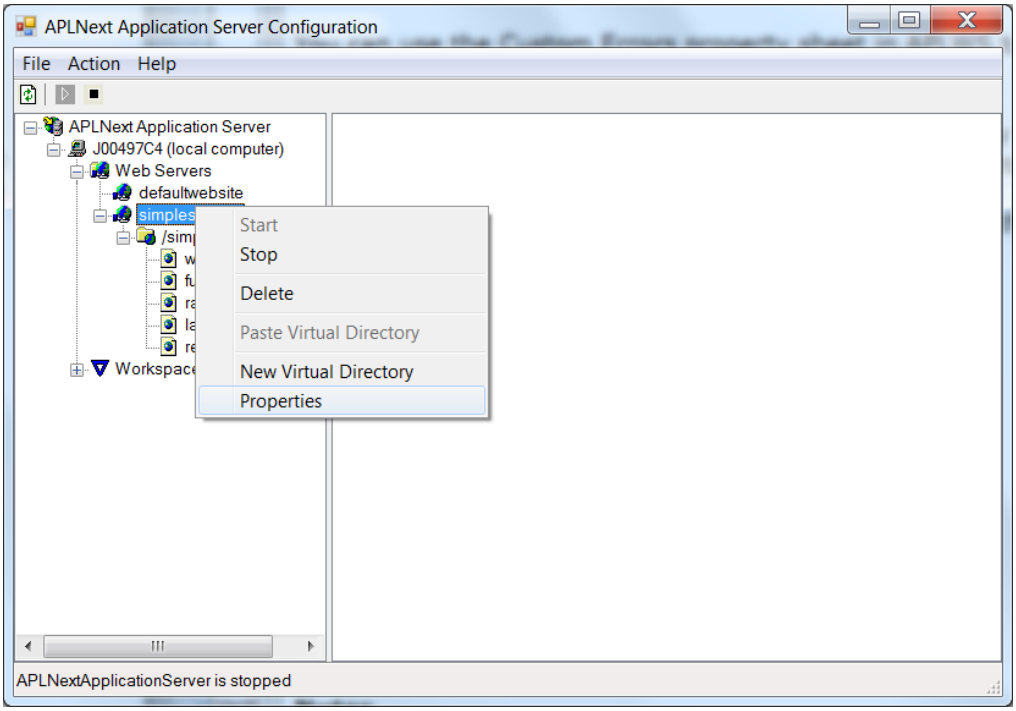
Add/delete a script filter using the subordinate dialog:



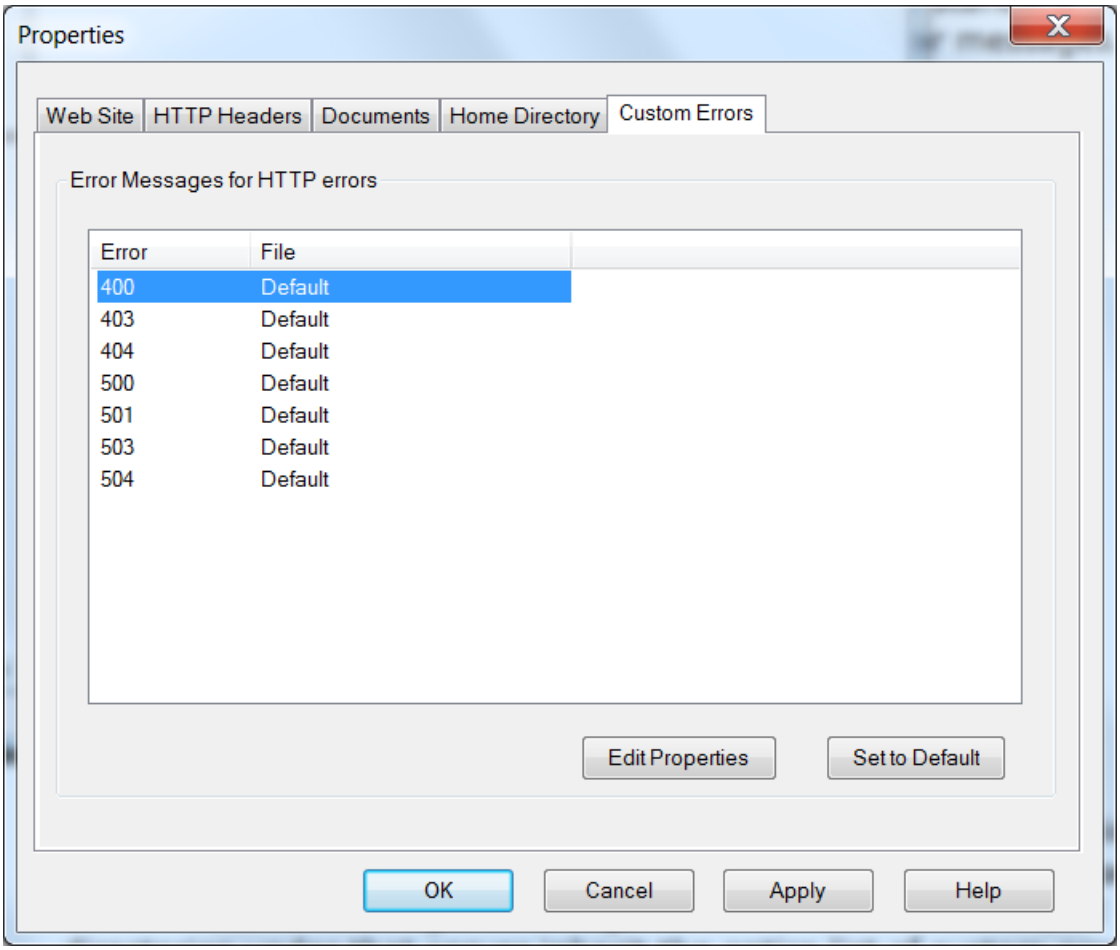
Custom Errors Property Sheet



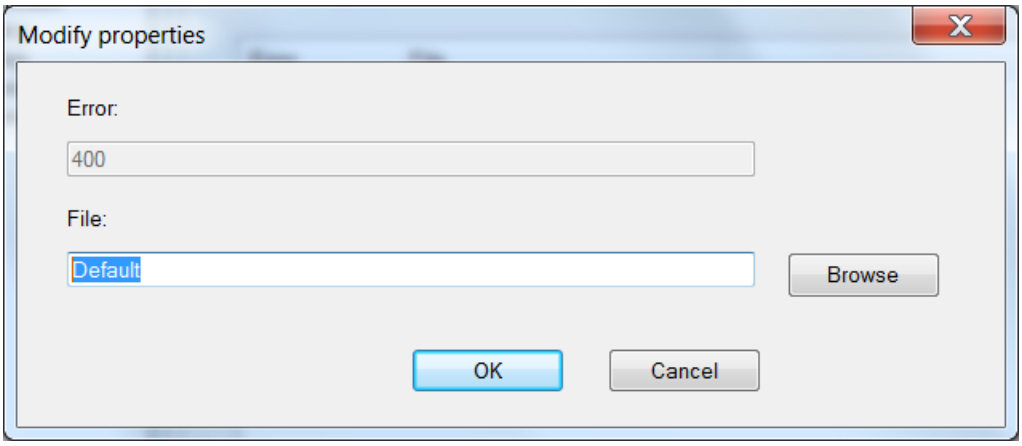
The AAS configuration can be modified to customize certain http error messages. To customize one of these http error messages, select the web service in the APLNextApplication Server Admin tool and use the context menu [right click] to select properties:



On the ‘Custom Errors’ tab, select one of the http error codes and click the ‘Edit Properties’ button:



In the ‘File:’ field enter the filename of the file containing the html-format custom error message.



The default http error messages for these codes are:

Error Code	Error Message
400	Bad request

403	Execute access forbidden
404	Not found
500	Internal server error
501	Not implemented
502	Bad gateway
503	Out of resources
504	Gateway timeout

Custom error messages appear as a list in the APLWS Admin which is treated as a single property by APLWS. For example, when a set of custom error messages is configured at the Web site level, all directories under that web service inherit the entire list of custom error messages.

Creating virtual paths within a web site

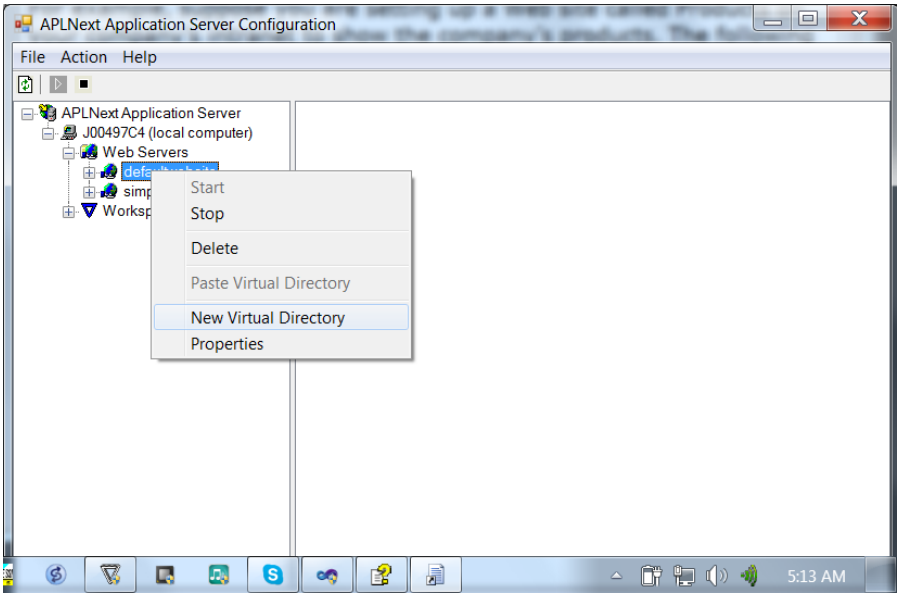
In AAS a web service is associated with an APL+Win function. Every such web service in a web site node has an associated virtual path [virtual directory]. Such a virtual path is an alias the APL+Win workspace on the server and the APL+Win function in that workspace that supports the web service.

The virtual path provides a web standards identifier for accessing the web service. For example if a web site needed to provide the client with access to a specific web service and that web service business rules and algorithms are implemented via an APL+Win function, the virtual path for that web service could be used by GUI controls on an html-format web page to invoke that web service.

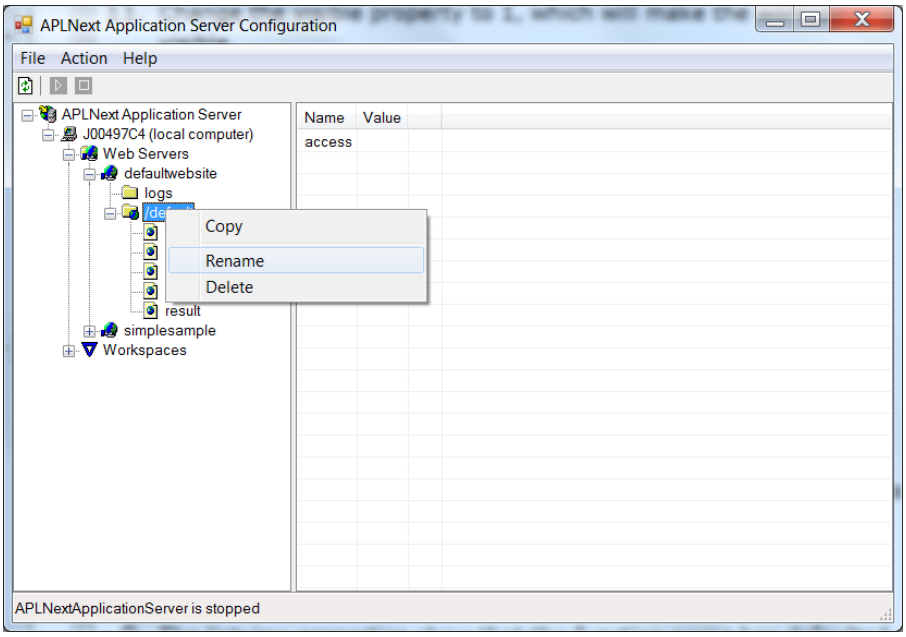
The following instructions are merely an illustrative example not designed to produce a working virtual path. The APL+Win Web Services forum contains a simple example of creating a virtual path associated with an APL+Win function which can be used to create a working example. The AAS installer also installs several working application system examples to the target machine.

To create a virtual path associated with an APL+Win function in an APL+Win workspace on the server machine:

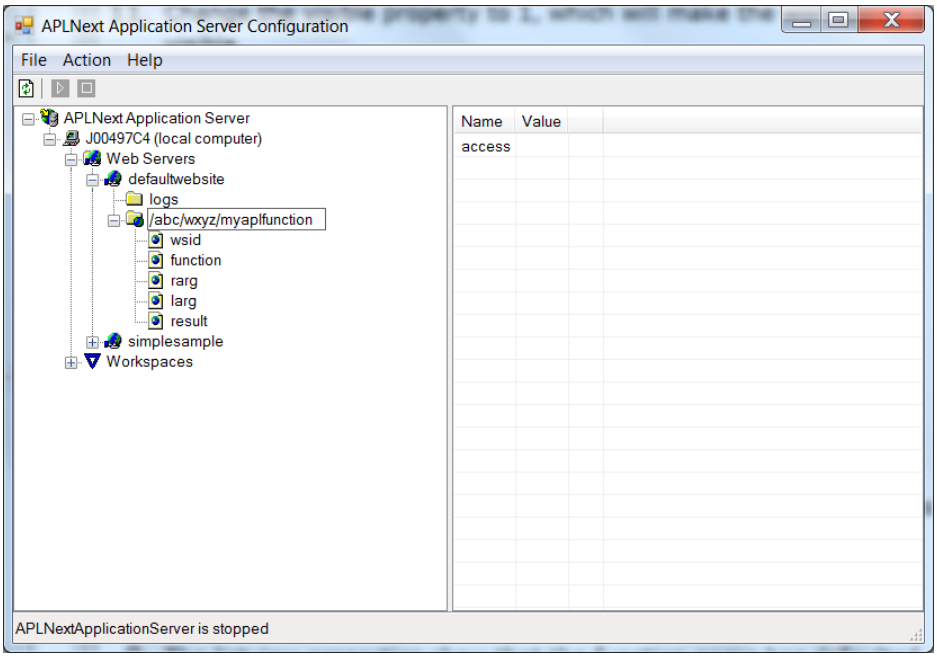
From the main form of the AAS administration tool select the web site to contain the virtual directory and use its context menu [right click] to select 'New Virtual Directory':



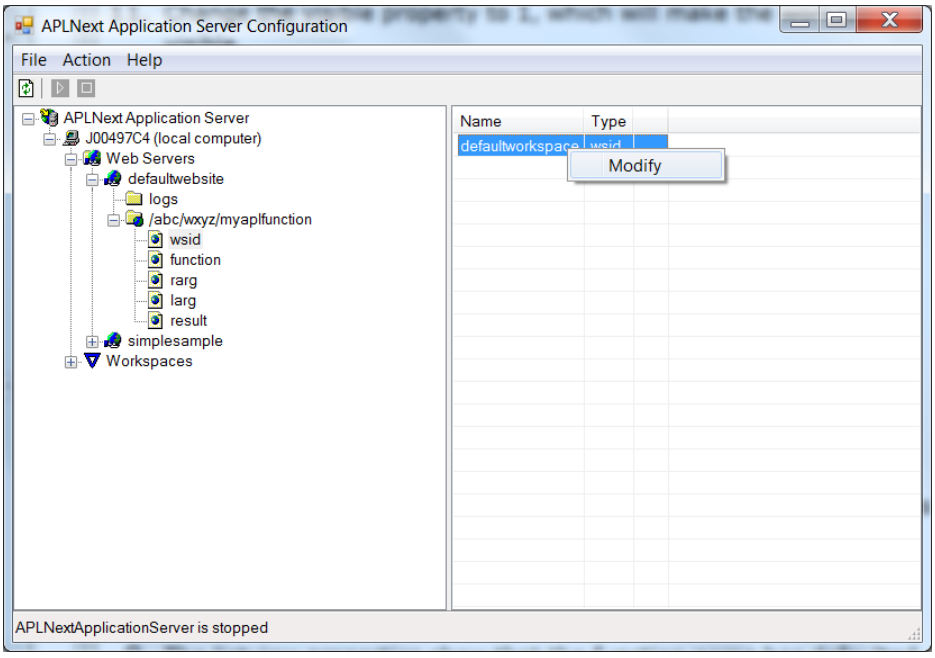
Open the web site node and select the '/default' virtual path name. Use its context menu [right click] to select the 'Rename' option:



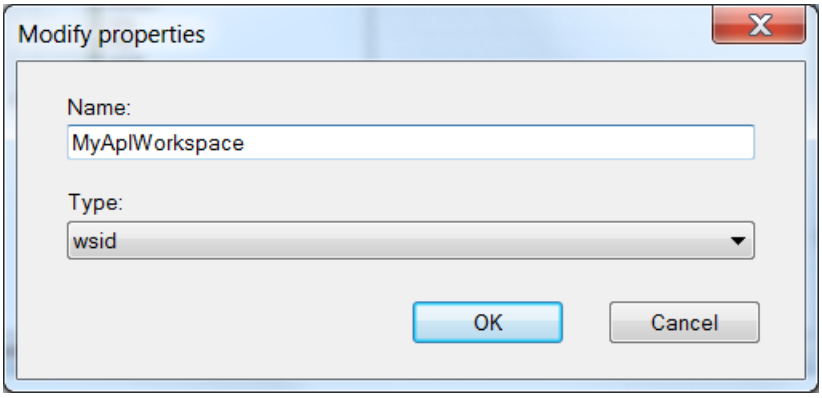
Rename the virtual path as desired, for example to '/abc/wxyz/myaplfunction'. This field value is not case sensitive.



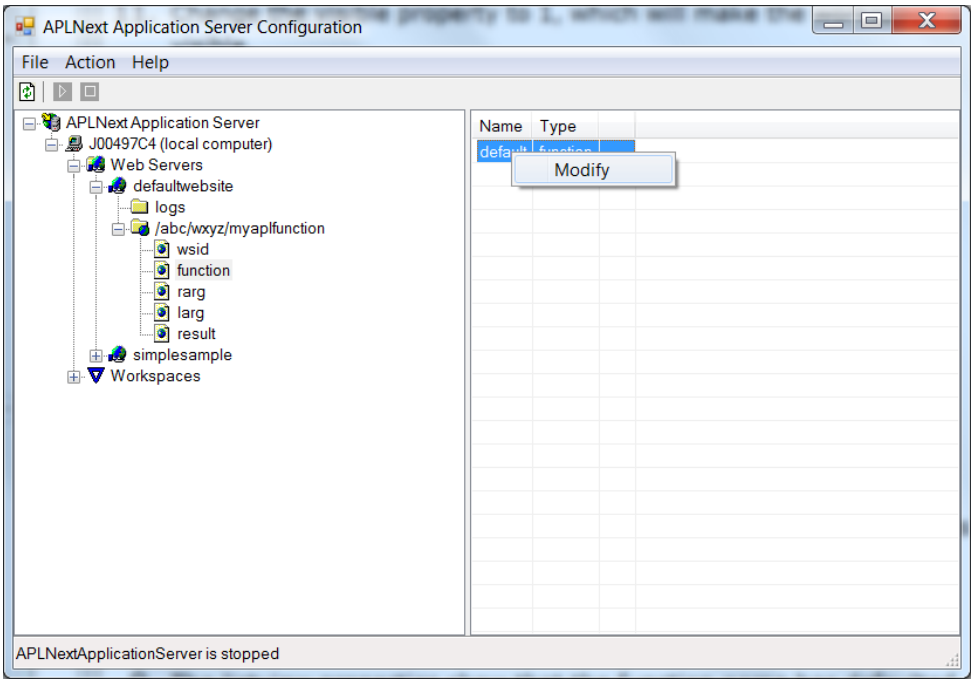
Select the 'wsid' node and use the context menu of the Name field to select the 'Modify' option:



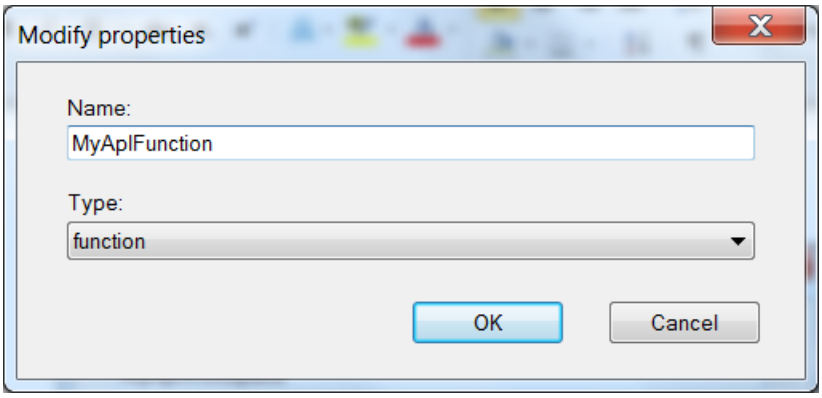
Change the 'defaultworkspace' name to the context-sensitive name of the APL+Win workspace containing the APL+Win function which will support this web service and click the OK button. If desired this workspace name can be an alias of the actual APL+Win workspace name.



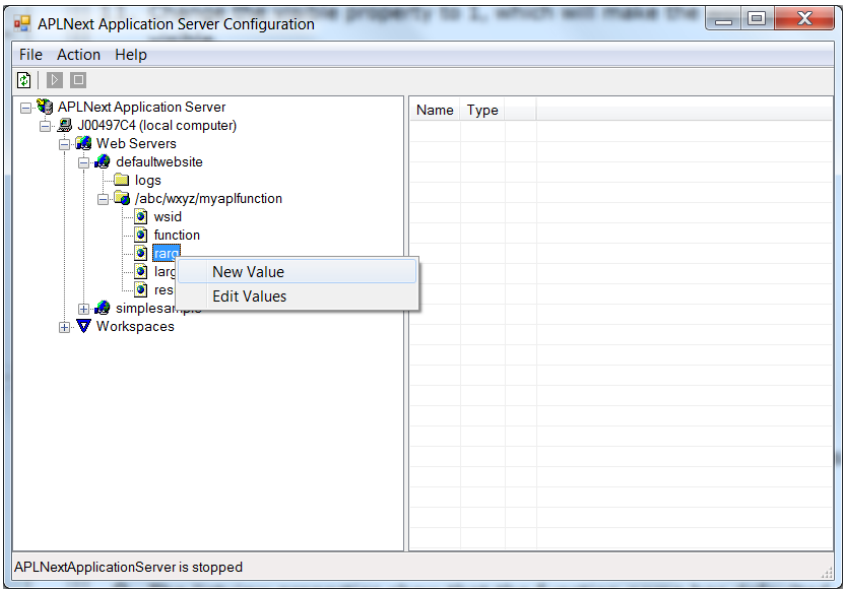
Select the ‘function’ node on the left and use the context menu of the ‘Name’ field to select the ‘Modify’ option:



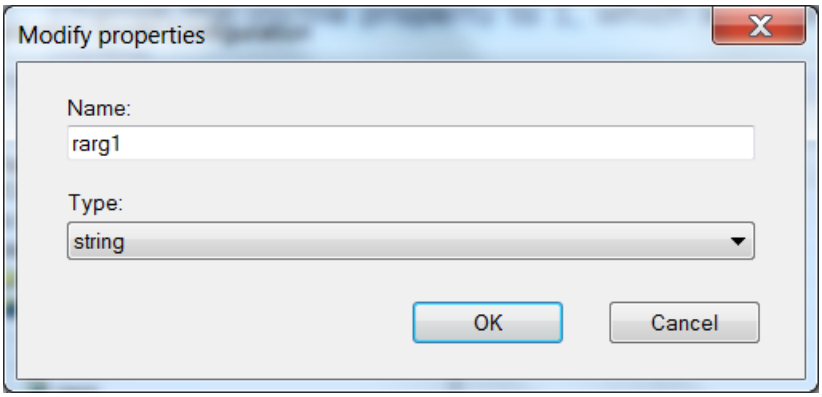
Enter the context-sensitive name of the APL+Win function which will support the web service and click the OK button:



Select the ‘rarg’ node on the left and use its context menu to select the ‘New Value’ option:

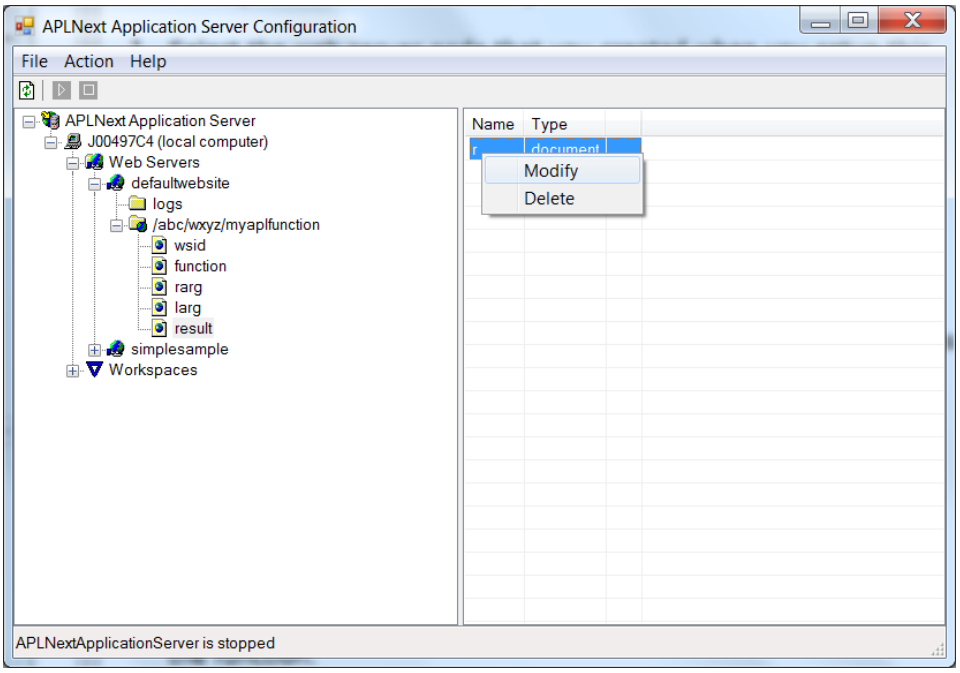


Enter the name of the (1st) right argument and select the applicable data type for the right argument. Selecting the applicable data type for a function argument is ‘tricky’ because it requires knowledge of the client data source and the available web standards argument data types supported by AAS.

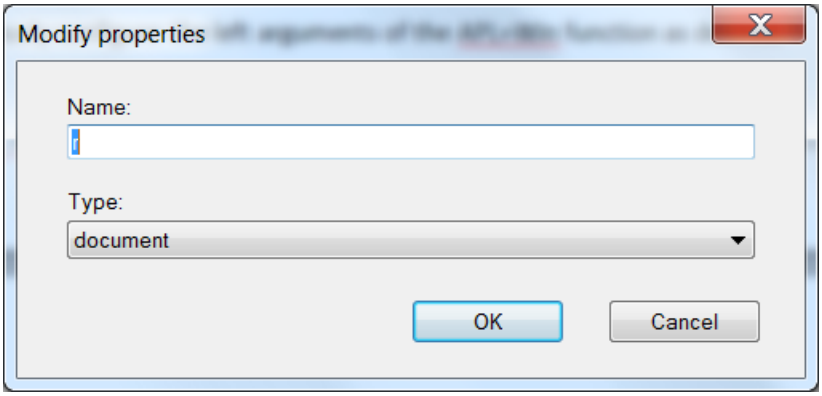


Analogously configure the left arguments of the APL+Win function as desired.

Every web service must have a result, so that the associated APL+Win function must have one or more explicit results. Configure the default result of the APL+Win function by selecting the ‘result’ node and using the context menu of the Name field to select the ‘Modify’ option.

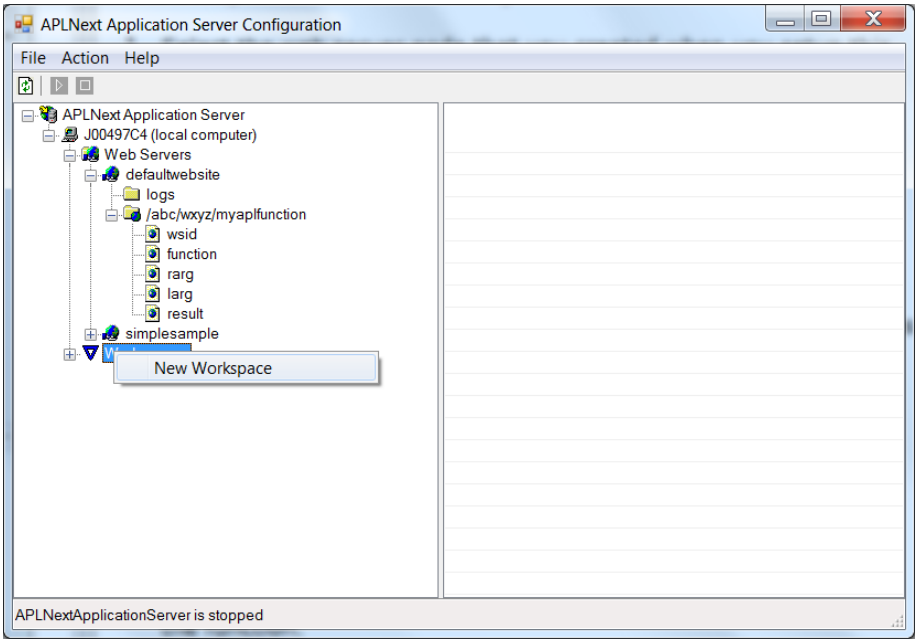


Enter the name of the result and select the appropriate result type. Selecting the applicable data type for a function result is also 'tricky' because it requires knowledge of the client's desired result type and the available web standards result data types supported by AAS.

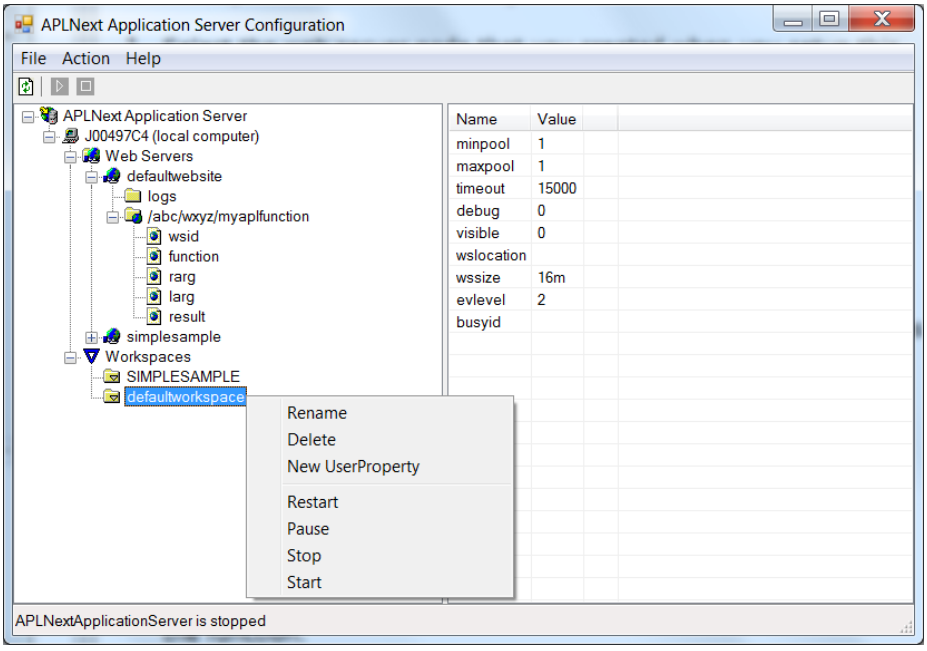


Because an APL+Win workspace can contain several functions which are exposed as web services, the configuration of the APL+Win workspace is distinct from the configuration of the APL+Win functions.

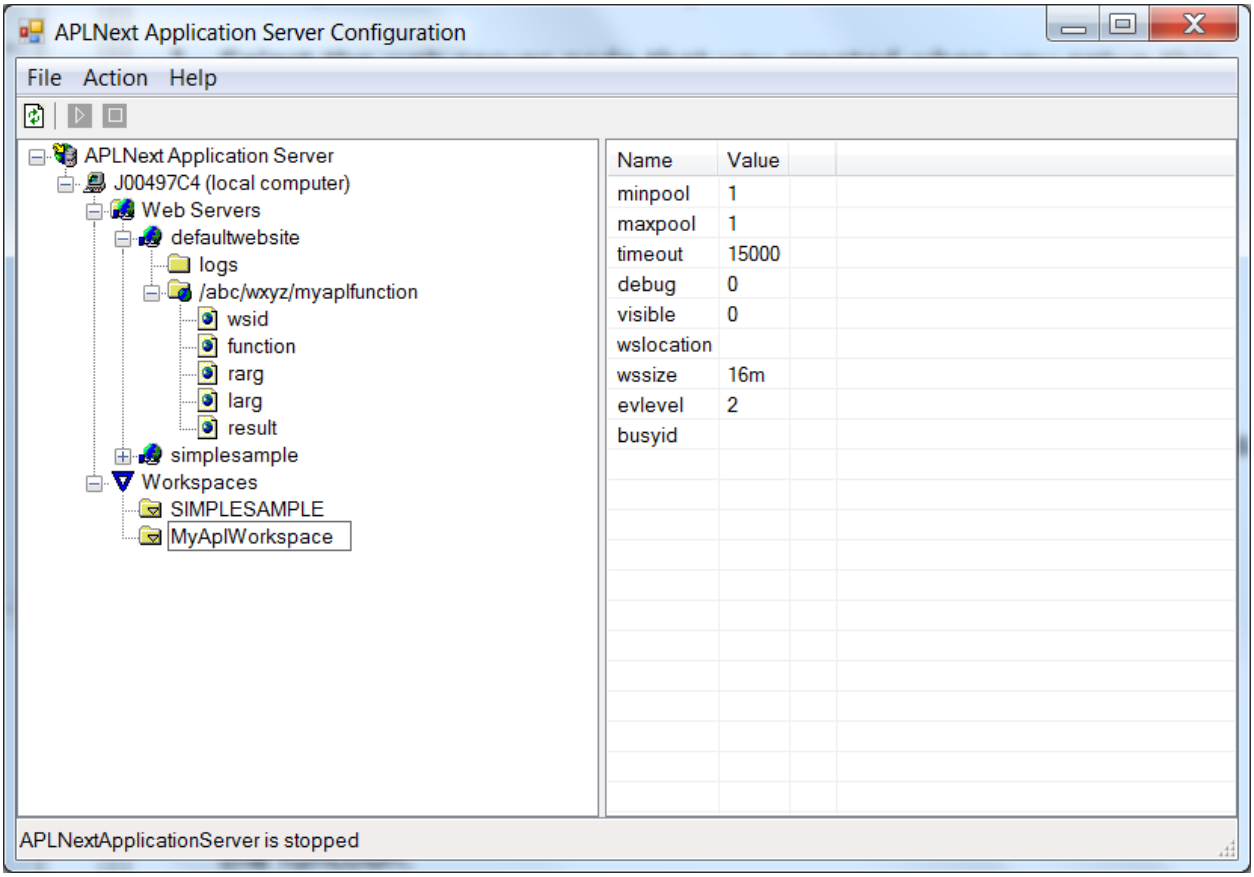
Select the 'Workspaces' node on the left of the main form of the AAS administration tool and use its context menu to select the 'New Workspace' option:



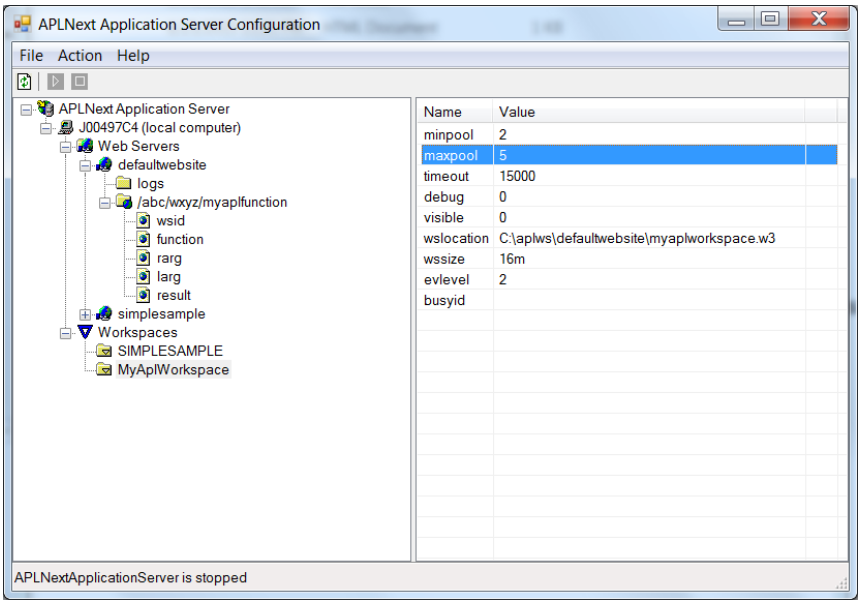
Open the 'Workspaces' node, select the 'defaultworkspace' node and use its context menu to select the 'Rename' option:



Enter the context-sensitive, actual or alias name of the APL+Win workspace, observing that it must exactly match the workspace name used in the configuration of the virtual directory of the APL+Win function exposed as a web service.



Using the context menu of each field in the workspace configuration, enter the appropriate values:



Administration – Workspace Properties

APL+Win workspaces contain APL+Win functions which perform the business rules and algorithm processing to satisfy a client request. The required APL+Win workspace properties must be properly configured in AAS for them to be accessible when a client request is received.

minpool & maxpool

The minpool and maxpool settings determine the number of simultaneous client requests for this web service which can be handled by AAS. Available machine resources can influence the values selected for these fields. These property values control the number of instances of APL+Win ActiveX engine that AAS will create on the server-side. Depending on the user load of the web service and the programmer-specified minpool and maxpool values, AAS will create or close these APL+Win ActiveX instances.

For minpool enter the minimum number of instances of APL+Win ActiveX engine which will load this workspace when this web service is started. A value of at least one is required. Increasing the value will mean that for the specified number of clients the workspace will have been pre-loaded and available to respond to those client requests.

For maxpool enter the maximum number of instances of APL+Win ActiveX engine which will load this workspace as user load on this web service varies. As user load on the web server increases AAS will instantiate additional APL+Win ActiveX engines and load this workspace. If the user load exceeds the available number of instances of APL+Win ActiveX engines, the requests will be queued by AAS.
timeout: Enter the timeout in milliseconds after which a client request will be cancelled by AAS. Timeouts are essential to maintain availability and responsiveness of a web service when user load increases.

debug

Enter 0 to have AAS consider the timeout setting or 1 to have AAS ignore the timeout setting. A setting of 1 here might cause a few long-running client requests to block access to the web service by other clients.

visible

Enter 0 to make the APL+Win ActiveX engine instances not visible. Because of Microsoft Windows security considerations, an entry of 1 here is effective only if APLNext Desktop Server is used which runs as a Windows application rather than APLNext Application Server which runs as a Windows service.

location

Enter the full physical address and name of the APL+Win workspace so that it can be loaded by the instances of APL+Win ActiveX engine created by AAS.

wssize

This field is reserved, enter 16m. The actual workspace size is controlled by the APL+Win 'ini' file.

evlevel

This field is reserved, enter 2.

busyid

This field supports the scaling of a web service. Enter the url of a different AAS web server which also supports this web service. Client requests which cannot be handled by the current web server are sent to the web server at the url in the 'busyid' field.

Web service? Web site?

A web service is any functionality which is supported by a web server and exposed to clients. The term web site generally refers to a particular type of web service which provides for server-side static or dynamic content using functionality incorporated in the server-side software which is accessed via a human-oriented GUI on the client-side.

Every web site is associated with a physical or virtual machine within a computer network that makes available information to authorized clients, generally via a web browser, e.g. Microsoft Internet Explorer, Apple Safari, Google Chrome, etc. The computer network may be an intranet, maintained for exclusive use by an enterprise or the public Internet. Information on the web site is generally published as html-format pages if it is to be viewed by a web browser. A web site will generally have a hierarchy of web pages starting with the home page or default document. The hierarchy of web pages is associated via links or GUI controls activated by client actions.

The dynamic functionality of a web site is usually supported by one or more web services. These web services of a web site can be based on APL+Win functions in APL+Win workspaces. The static content of a web site can be dynamically created by APL+Win programmer-developed functions or created in advance using html authoring tools and stored in the home directory.

Quick Sample Static Content Web Site Setup

AAS does not create a default web service when it is installed because there are many types of web services possible which may be designed and implemented by the programmer. In this section a very simple web site web service will be created using AAS.

Create the home page for the web site

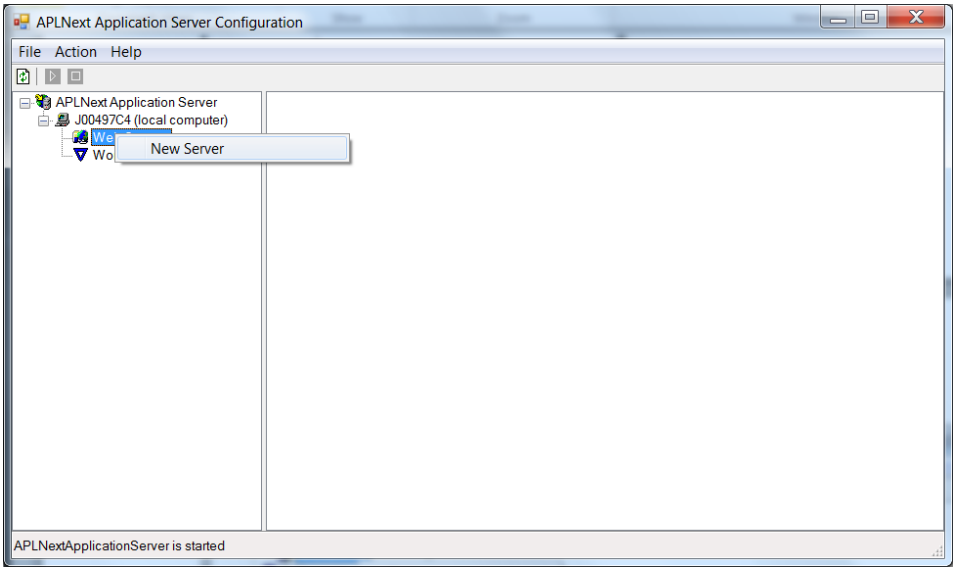
Using Microsoft Notepad create a new text document with file name 'default.html' with the following content and save it to a directory on the target workstation called 'c:\aplws\defaultwebsite\'. This directory will be the home directory for the new web site.

```
<!DOCTYPE html>
<html>
<head>
<title>Home Page Title</title>
```

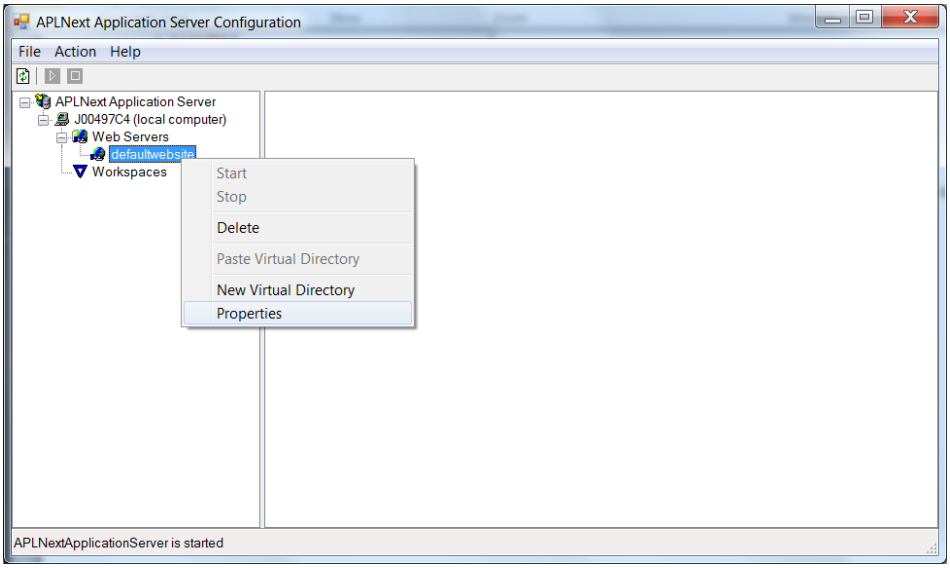
```
</head>
<body>
<h1>Home Page</h1>
<p></p>
<p>Welcome to the Home Page</p>
</body>
</html>
```

Configure AAS for the new web site

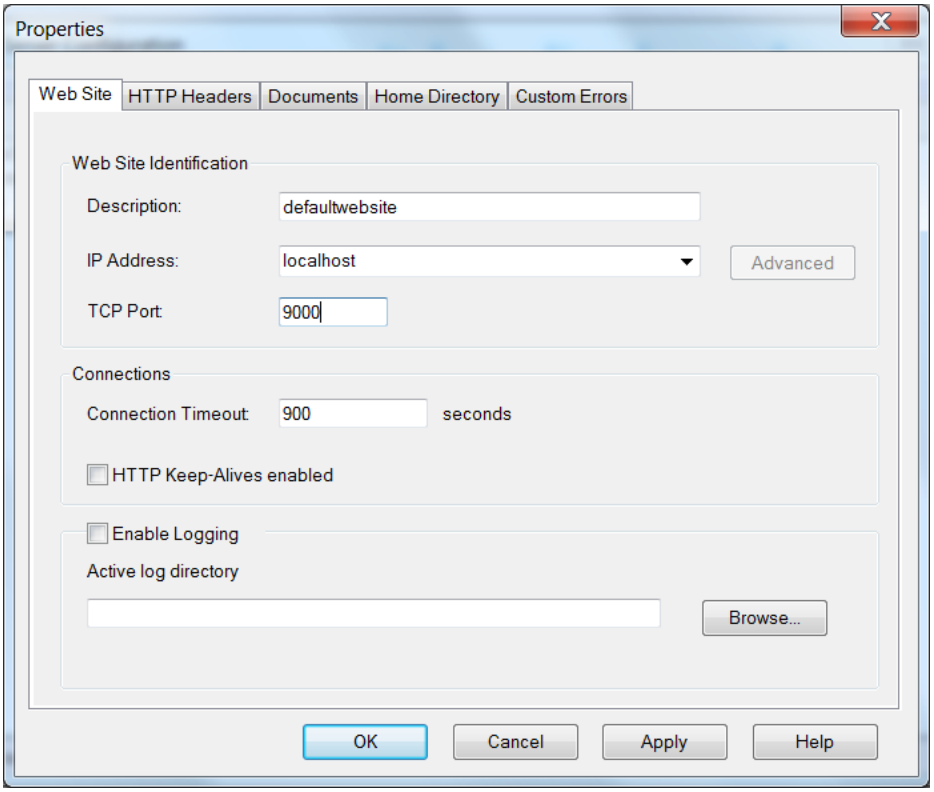
- Start an instance of the APLNext Application Server Admin tool
- Open the server tree on the left side of the dialogue and select the Web Servers node.
- Using the Action > New > Server menu item create a new web site which will automatically be called ‘defaultwebsite’.



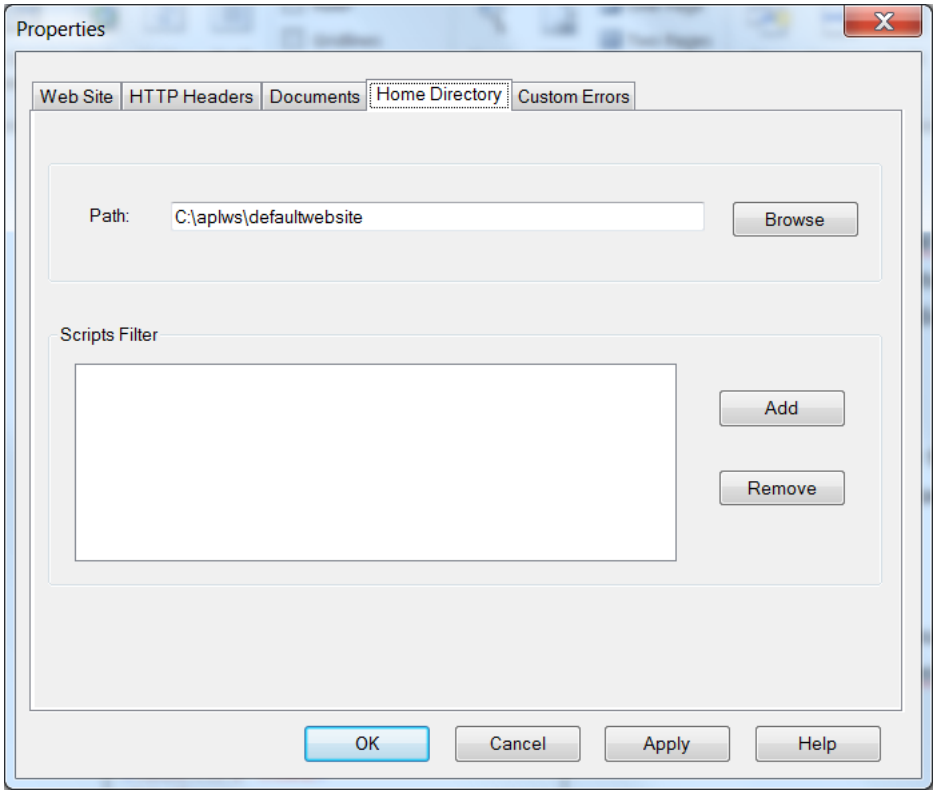
- Use the context menu [right click] for the defaultwebsite to select Properties



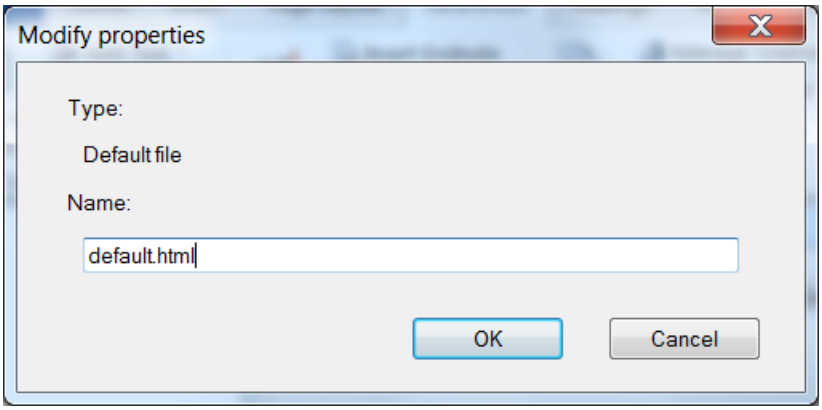
- Complete the entries for the Web Site tab of the Properties dialog as follows and click the Apply button. In a production environment the ip address or server name of the server would be used instead of 'localhost'. The 'localhost' ip address is often used by programmers to test web services on their workstation. The '9000' port number was selected for this example to avoid interference with any other web servers which might be operating on the target machine. In a production environment a port number would probably be selected according to Internet conventions.

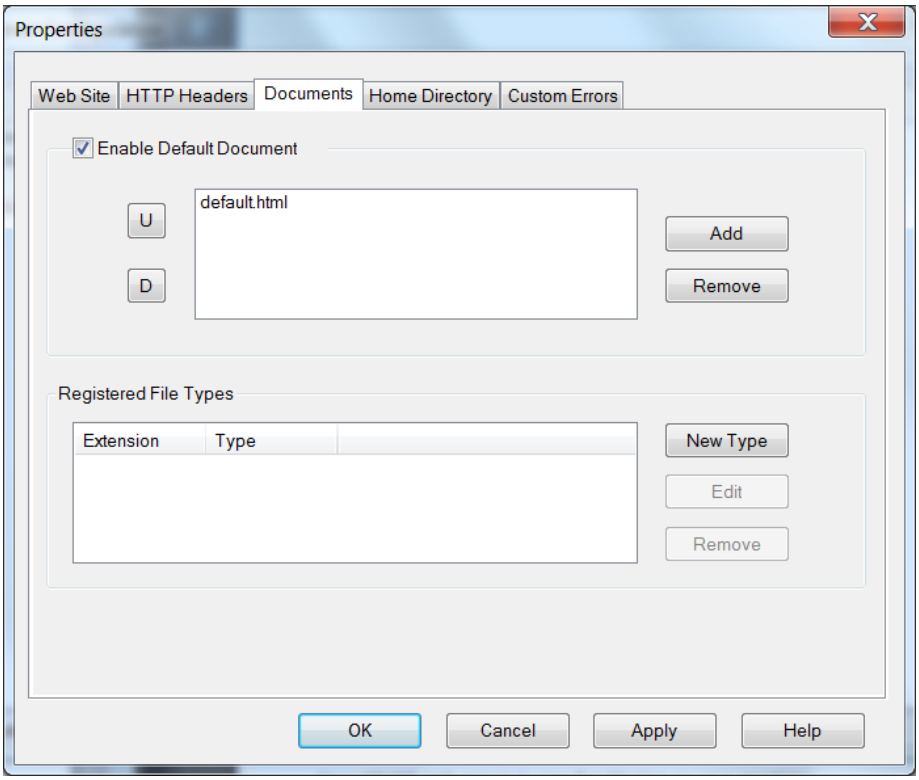


- Complete the entries on the Home Directory tab as follows and click the apply button:

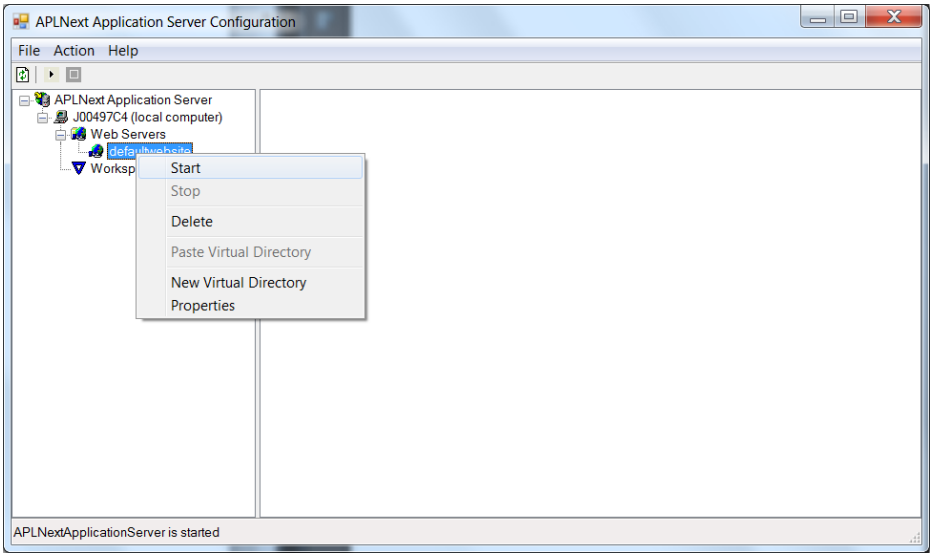


- Complete the entries on the Documents tab as follows and click the apply button. To complete these entries click the Add button, enter 'default.html' as the document name and click the OK button. When the 'default.html' file name is in the Documents tab field, check the Enable Default Document check box and click the Apply button.

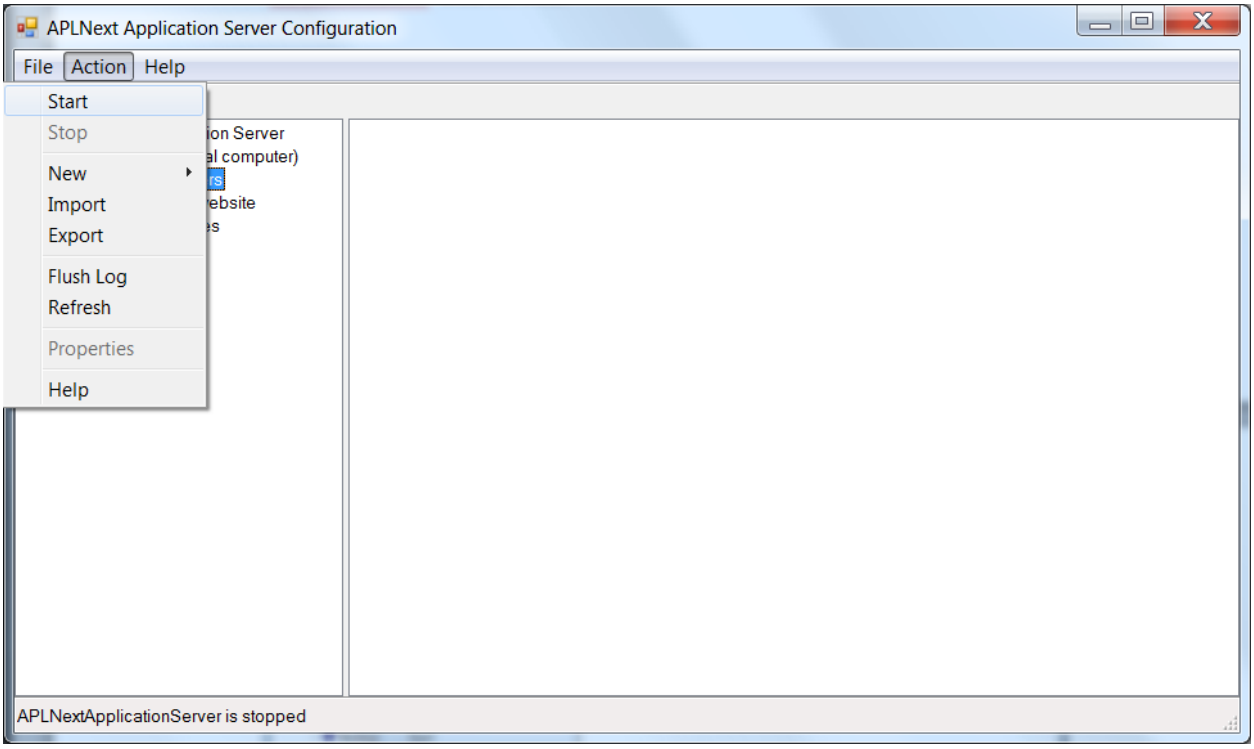




- Click the OK button to return to the main APLNext Application Server Admin dialog.
- Use the context menu [right click' on the 'defaultwebsite' node and select 'Start' to start the 'defaultwebsite' web service.



- If necessary start the AAS web server by selecting the 'Web Servers' node and using the Action > Start menu item.



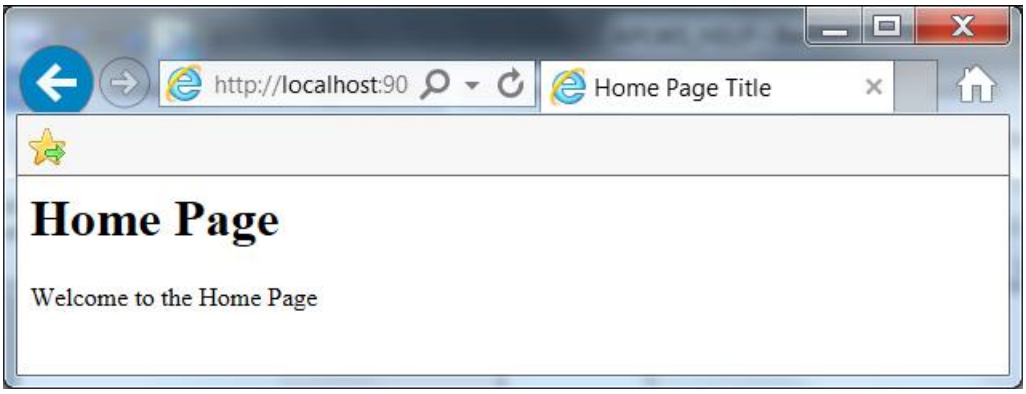
- Notice that both the specific 'defaultwebsite' web service must be started and the AAS web server must be started to enable client use of the 'defaultwebsite' web service.

Test the new web site

Open an instance of a web browser [Microsoft Internet Explorer in this screen capture] and enter the url of the default web site: 'http://localhost:9000/' and use the Enter key to display the web site's home page.

The physical location of the home directory is not directly accessed by a web service client. Instead the AAS web server associates a physical directory containing web site content with a virtual path. In this case the virtual path is very simple, i.e. an empty virtual path.

When this web page is displayed in the client's browser it indicates that AAS has been correctly configured and started to enable the 'defaultwebsite' web service.



Expose an APL+Win function as a web service

The preceding example illustrated a typical web site providing clients with static content as html-format documents. A web site which interacts with the client to provide dynamic content using APL+Win functions to support the business rules and algorithms is also easy to implement using AAS.

Generally any APL+Win function can be exposed as a web service. In most cases that function will have an argument containing the required input provided by the web service client and will have a result that represents the response of the APL+Win function to the client request. The function result must be carefully constructed so that is represents an http response which is compatible with the client.

Since the APL+Win function exposes as a web service is running on a server it cannot directly present a server-side user interface to the client. Generally the user interface for a web service designed for a human is browser-based and written in html. Such a user interface will have GUI controls to collect client-provided input to the APL+Win function as well as GUI controls to initiate the transmission of the client request to AAS.

For web services which are accessed by other machines, the access to the functionality of the web service is generally through http-formatted requests under program control. The associated APL+Win function will have an argument to contain the request input data and a result to contain the response data.

Formatting the input and output of the associated APL+Win function between the web standards format and the APL+Win format is done automatically by AAS. Optionally an APL+Win function may receive and directly use the client request in web standards format or prepare the result directly in web standards format.

The APL2000 Web Services forum contains a pdf-format document with the details of exposing an APL+Win function as an AAS web service.

APL+Win Function Right and Left Argument Configuration Options

APL+Win functions may have multiple right and left arguments. Each such argument requires a separate configuration name and type. When one of these types is selected, AAS assumes that the client request will contain an argument of this type and AAS will provide the associated APL+Win function argument element with this data type from the client. If the web standards data type is an array, the row and column order will be the inverse of the row and column order in APL+Win so the □dtr system function or the ⍉ function will be useful.

Entity-body

The entity body (if any) sent with an HTTP request or response is in a format and encoding defined by the Entity-Header fields. Entity-Body = *OCTET

An entity body is included with a request message only when the request method calls for one. The presence of an entity body in a request is signaled by the inclusion of a Content-Length header field in the request message headers. HTTP/1.0 requests containing an entity body must include a valid Content-Length header field.

For response messages, whether or not an entity body is included with a message is dependent on both the request method and the response code. All responses to the HEAD request method must not include a body, even though the presence of entity header fields may lead one to believe they do. All 1xx (informational), 204 (no content), and 304 (not modified) responses must not include a body. All other responses must include an entity body or a Content-Length header field defined with a value of zero (0).

When an Entity-Body is included with a message, the data type of that body is determined via the header fields Content-Type and Content-Encoding. These define a two-layer, ordered encoding model: entity-body = Content-Encoding(Content-Type(data))

A Content-Type specifies the media type of the underlying data. A Content-Encoding may be used to indicate any additional content coding applied to the type, usually for the purpose of data compression, that is a property of the resource requested. The default for the content encoding is none (i.e., the identity function).

Any HTTP/1.0 message containing an entity body should include a Content-Type header field defining the media type of that body. If and **only if** the media type is not given by a Content-Type header, as is the case for Simple-Response messages, the recipient may attempt to guess the media type via inspection of its content and/or the name extension(s) of the URL used to identify the resource. If the media type remains unknown, the recipient should treat it as type "application/octet-stream ".

When an Entity-Body is included with a message, the length of that body may be determined in one of two ways. If a Content-Length header field is present, its value in bytes represents the length of the Entity-Body. Otherwise, the body length is determined by the closing of the connection by the server.

Closing the connection cannot be used to indicate the end of a request body, since it leaves no possibility for the server to send back a response. Therefore, HTTP/1.0 requests containing an entity

body must include a valid Content-Length header field. If a request contains an entity body and Content-Length is not specified, and the server does not recognize or cannot calculate the length from other fields, then the server should send a 400 (bad request) response.

Binarywrapl

This argument type serializes any APL+Win variable value on the client-side and provides it to the associated APL+Win server-side function as that APL+Win variable value. This argument type is useful when the client is an APL+Win function accessing a server-side APL+Win function associated with an AAS web service.

Client-ip

This is the internet protocol address of the client.

Cookie

The value of this argument type is the cookie content set by the return value of the previous client request.

Entity-body-decoded

Client input data from an html-format form document will be returned as a 2-row matrix with one column for each client input element with the 1st column the input element name and the 2nd element the input element value.

Entity-body-filename

The server-side filename containing the uploaded client input.

Entity-body-utf8

The client input in entity-body format using the Unicode UTF-8 coding.

Float

An IEEE floating point number

Header

This value is the request header in web standard format.

Header-parsed

The request header returns as a 2-row matrix with one column for each field in the header with the 1st column the field name and the 2nd column the field value.

If-modified-since

The If-Modified-Since request-header field is used with the GET method to make it conditional. If the requested resource has not been modified since the time specified in this field, a copy of the resource

will not be returned from the server; instead, a 304 (not modified) response will be returned without any Entity-Body. If-Modified-Since = "If-Modified-Since" ":" HTTP-date

An example of the field is: If-Modified-Since: Sat, 29 Oct 1994 19:43:31 GMT

A conditional GET method requests that the identified resource be transferred only if it has been modified since the date given by the If-Modified-Since header. The algorithm for determining this includes the following cases:

- If the request would normally result in anything other than a 200 (ok) status, or if the passed If-Modified-Since date is invalid, the response is exactly the same as for a normal GET. A date which is later than the server's current time is invalid.

- If the resource has been modified since the If-Modified-Since date, the response is exactly the same as for a normal GET.

- If the resource has not been modified since a valid If-Modified-Since date, the server shall return a 304 (not modified) response.

The purpose of this feature is to allow efficient updates of cached information with a minimum amount of transaction overhead.

Int

A 32-bit integer

Method

The method token indicates the method to be performed on the resource identified by the Request-URI. The method is case-sensitive. The list of methods acceptable by a specific resource can change dynamically. The client is notified through the return code of the response if a method is not allowed on a resource. Servers should return the status code 501 (not implemented) if the method is unrecognized or not implemented.

Process-id

This is the unique positive integer id assigned by AAS to each APL+Win ActiveX engine instance created by AAS. This argument is useful for certain log-in and load balancing techniques.

Publichttpdir

This is the web service home directory which can be used to determine the location of the publicly-available content.

Referrer

The Referer request-header field allows the client to specify, for the server's benefit, the address (URI) of the resource from which the Request-URI was obtained. This allows a server to generate lists of back-links to resources for interest, logging, optimized caching, etc. It also allows obsolete or mistyped links

to be traced for maintenance. The Referer field must not be sent if the Request-URI was obtained from a source that does not have its own URI, such as input from the user keyboard.

Referer = "Referer" ":" (absoluteURI | relativeURI)

Example:

Referer: http://www.w3.org/hypertext/DataSources/Overview.html

If a partial URI is given, it should be interpreted relative to the Request-URI. The URI must not include a fragment.

Request-uri

The Request-URI is a Uniform Resource Identifier that identifies the resource upon which to apply the request. Request-URI = absoluteURI | abs_path

The two options for Request-URI are dependent on the nature of the request.

The absoluteURI form is only allowed when the request is being made to a proxy. The proxy is requested to forward the request and return the response. If the request is GET or HEAD and a prior response is cached, the proxy may use the cached message if it passes any restrictions in the Expires header field.

Note that the proxy may forward the request on to another proxy or directly to the server specified by the absoluteURI. In order to avoid request loops, a proxy must be able to recognize all of its server names, including any aliases, local variations, and the numeric IP address. An example Request-Line would be: GET /TheProject.html HTTP/1.0

The most common form of Request-URI is that used to identify a resource on an origin server or gateway. In this case, only the absolute path of the URI is transmitted (see Section 3.2.1, abs_path). For example, a client wishing to retrieve the resource above directly from the origin server would create a TCP connection to port 80 of the host "www.w3.org" and send the line:

GET /pub/WWW/TheProject.html HTTP/1.0

This line is then followed by the remainder of the full request. Note that the absolute path cannot be empty; if none is present in the original URI, it must be given as "/" (the server root).

The Request-URI is transmitted as an encoded string, where some characters may be escaped using the "% HEX HEX" encoding defined by RFC 1738 [4]. The origin server must decode the Request-URI in order to properly interpret the request.

Serverid

This value is the name of the server which received the request. Using this value it is possible to determine the AAS server which received the request even if several different servers access the same APL+Win function in the same APL+Win workspace.

Soapwrapl

The client-side value will be serialized into a web standards SOAP text string. AAS will de-serialize the value and provide it to the associated APL+Win function in APL+Win format. This option is useful if the client is an APL+Win function.

String

The client-side value is provided to APL+Win as a character vector.

User-agent

The User-Agent request-header field contains information about the user agent originating the request.

This is for statistical purposes, the tracing of protocol violations, and automated recognition of user agents for the sake of tailoring responses to avoid particular user agent limitations. Although it is not required, user agents should include this field with requests. The field can contain multiple product tokens and comments identifying the agent and any sub-products which form a significant part of the user agent. By convention, the product tokens are listed in order of their significance for identifying the application. User-Agent = "User-Agent" ":" 1*(product | comment)

Example: User-Agent: CERN-LineMode/2.15 libwww/2.17b3

Some current proxy applications append their product information to the list in the User-Agent field. This is not recommended, since it makes machine interpretation of these fields ambiguous. Some existing clients fail to restrict themselves to the product token syntax within the User-Agent field.

APL+Win Function Result Configuration Options

AAS supports the following result types. When one of these types is selected, it is expected by AAS that the associated APL+Win function's result element will satisfy the requirements of this type and AAS will provide this data type to the client as part of the response.

Apl2binarywrapl

This option is a serialization of an APL+Win variable value to a string of characters which can be transmitted via a web service back to the client and deserialized there. This argument type is useful when the client is an APL+Win function accessing a server-side APL+Win function associated with an AAS web service.

Apl2soapwrapl

This option is a soap envelope serialization of an APL+Win value to a string of characters which can be transmitted via a web service back to the client and deserialized there. This argument type is useful when the client is an APL+Win function accessing a server-side APL+Win function associated with an AAS web service. Other clients besides APL+Win can soap serialize 'structured' data, transmit it to AAS and receive a response which is soap serialized.

Content-disposition

This option is used to add a content disposition header to the result of the web service. Such a content disposition can indicate how it should be presented to the client or the disposition of the server-side file containing the content of the result.

Content-type

Many such types are available, for example 'application/vnd.ms.excel' for a Microsoft Excel file or 'application/pdf' for an Adobe pdf-format document.

Cookie

The cookie content is saved in the web browser sandbox on the client machine.

One-to-one

By default http requests are stateless, i.e. each request is handled independently, so that the same APL+Win ActiveX engine instance many not handle consecutive requests from the same client.

Sometimes it is advantageous to have consecutive requests from the client handled by the same APL+Win ActiveX engine instance because it takes several client requests to establish the desired workspace variable values. Having a 'stateful' client and server relationship means that the state of the APL+Win ActiveX engine instance will not be modified by any other client while the 'one-to-one' status exists. The 'one-to-one' relationship between client and web server support this feature. If the 'one-to-one' result value is 1 this establishes a 'stateful' association between the client and server. If the 'one-to-one' result value is 0 this ends the 'one-to-one' relationship between the client and server.

Status-code

The status-code element is a 3-digit integer result code of the attempt to understand and satisfy the request. The Status-Code is intended for use by automata. The first digit of the Status-Code defines the class of response. The last two digits do not have any categorization role.

HTTP status codes are extensible. HTTP applications are not required to understand the meaning of all registered status codes, though such understanding is obviously desirable. However, applications must understand the class of any status code, as indicated by the first digit, and treat any unrecognized response as being equivalent to the x00 status code of that class, with the exception that an unrecognized response must not be cached. For example, if an unrecognized status code of 431 is received by the client, it can safely assume that there was something wrong with its request and treat the response as if it had received a 400 status code. In such cases, user agents should present to the user the entity returned with the response, since that entity is likely to include human-readable information which will explain the unusual status.

Reason-phrase

The Reason-Phrase is intended to give a short textual description of the Status-Code. The Reason-Phrase is intended for the human user. The client is not required to examine or display the Reason-Phrase.

Document

This is a text string formatted as an html-document, e.g. in the simplest form:

```
<html>
<body>my response document
</body>
</html>
```

Document-filename

The physical filename of a server-side file containing the response data. If the response data is very large or will remain valid for a significant period of time, this is a useful option.

Document-filename-delete

Indicates if the 'document-filename' should be deleted on the server after it has been received by the client: 0/No 1/Yes.

Expires

The expires http response header field provides the timestamp indicating when the returned content is considered obsolete, for example: Expires: Thu, 01 Dec 1994 16:00:00 GMT

Float

An IEEE floating point number.

Int

A 32-bit integer number.

Location

The location is response information provided by the server as a header field in conjunction with the status – code.

Progress-bar-info

Reserved.

Soap-body

The content serialized using soap.

Soap-envelope-start

The location in the http response of the beginning of the soap envelope.

Soap-envelope-end

The location in the http response of the end of the soap envelope.

String

A character string

Actions APL+Win may take to affect the web service

APL+Win functions are used to support the business rules and algorithms of a web service. These APL+Win functions are executed under the control of an instance of the APL+Win ActiveX server which has been instantiated by AAS.

Sometimes a server-side APL+Win function running in an instance of APL+Win ActiveX engine which was created by AAS will need to take action to affect the web service which it supports by sending a message to the AAS web server. The APL+Win 'Notify' method may be used for this purpose.

AAS is the client of these instances of APL+Win ActiveX engine and AAS has subscribed to the 'onNotify' event which fires when an APL+Win function uses the 'Notify' method. AAS incorporates 'onNotify' event handlers which take appropriate action based on the 'Notify' method arguments provided by the APL+Win function.

The syntax of the 'Notify' method when messaging AAS is:

```
Result←'#'⎕wi 'Notify' xmlArg
```

'xmlArg' is an xml-formatted text string that depends on the message to be sent and 'Result' is the response, if any, from AAS to the APL+Win function which used the 'Notify' method. The format of 'xmlArg' is:

```
<aplresult>
<aplmethod>methodName</aplmethod>
…
</aplresult>
```

'methodName' is selected from the available options and is not case sensitive.
'…' indicates additional parameters depending on the 'methodName'.

The Result, if any, is a two element APL+Win vector. The 1st element is a numeric scalar indicating if any error occurred when the AAS event handler for the 'onNotify' event was executed. A value of zero indicates no error. Any other value indicates an error, in which case the 2nd element of the Result will be a text string containing an error message, e.g '-1' 'Unknown Method'. If there is no error the 2nd element structure depends on the 'Notify' method argument.

POST

This option enables the APL+Win function to make an http 'POST' request, which generally provides data to the target web service which is used to alter the state of the application supported by that web service.

xmlArg for POST option
<aplresult> <aplmethod>POST</aplmethod> <apldata>entityBody</apldata> <aplurl>targetUrl</aplurl> <aplcookie>cookieContent</aplcookie> <apltimeout>timeoutSeconds</apltimeout> <aplcontent-type>contentType</aplcontent-type> </aplresult>

entityBody generally contains the name and value pairs of the data to be sent to the targetUrl

targetUrl is the url of the web service which is the target of this http post request

cookieContent is the information which will be stored in the cookie on the client's web browser sandbox as a result of this http post request. A cookie is a way to create stateful sessions with HTTP requests and responses by storing a small amount of data on the client. This context might be used to create, for example, a "shopping cart", in which user selections can be aggregated before purchase, or a magazine browsing system, in which a user's previous reading affects which offerings are presented.

timeoutSeconds is the number of seconds that the APL+Win function will wait to receive a response to the http post request before timing out.

contentType indicates the type of the http post request, for example 'text/html', 'image/png', 'image/gif', 'video/mpeg', 'text/css'.

Result: 0 (2-element array)
The 1st element of the result array is the response header

The 2nd element of the result is the name of the file on the server which contains the result of the http post request. AAS will not delete this file, so the programmer must incorporate this deletion, if appropriate, within the APL+Win function which uses this option.

GET

This option enables the APL+Win function to make an http 'GET' request which generally provides data to the target web service which is used to select applicable response data, but not modify the state of the application system supported by that web service.

xmlArg for GET option
<aplresult> <aplmethod>GET</aplmethod> <apldata></apldata> <aplurl>targetUrl</aplurl> <aplcookie>cookieContent</aplcookie> <apltimeout>timeoutSeconds</apltimeout> <aplcontent-type>contentType</aplcontent-type> </aplresult>

targetUrl is the url of the web service which is the target of this http get request which generally contains the name and value pairs of the data to be sent to the target server.

cookieContent is the information which will be stored in the cookie on the client's web browser sandbox as a result of this http get request. A cookie is a way to create stateful sessions with HTTP requests and responses by storing a small amount of data on the client. This context might be used to create, for example, a "shopping cart", in which user selections can be aggregated before purchase, or a magazine browsing system, in which a user's previous reading affects which offerings are presented.

timeoutSeconds is the number of seconds that the APL+Win function will wait to receive a response to the http get request before timing out.

contentType indicates the type of the http get request, for example 'text/html', 'image/png', 'image/gif', 'video/mpeg', 'text/css'.
Result: 0 (2-element array)
The 1st element of the result array is the response header

The 2nd element of the result is the name of the file on the server which contains the result of the http get request. AAS will not delete this file, so the programmer must incorporate this deletion, if appropriate, within the APL+Win function which uses this option.

APL

This option enables the APL+Win function to execute an APL+Win function which is included in the configuration of AAS on the target machine.

xmlArg for APL option
<aplresult> <aplmethod>apl</aplmethod> <aplurl>wsId</aplurl> <aplfunction>fnName</aplfunction> <apllarg>leftArg</apllarg> <aplrarg>rightArg</aplrarg> </aplresult>

wsId is the name of the APL+Win workspace included in the AAS configuration

fnName is the name of the APL+Win function included in the AAS configuration

leftArg is the left argument to the target APL+Win function.

rightArg is the right argument to the target APL+Win function

Result: 0 targetFunctionResult

Error

This option enables the APL+Win function to signal an unrecoverable error in its processing with a message to the client and close the instance of APL+Win ActiveX engine which is executing the function.

xmlArg for Error option
<aplresult> <aplmethod>error</aplmethod> <apldata>errorMessage}</apldata> </aplresult>

errorMessage is the text string containing the desired error message for the client.

Result: 0 ''

Kill

This option enables the APL+Win function to close the instance of APL+Win ActiveX engine which is executing the function.

xmlArg for Kill option
<aplresult> <aplmethod>kill</aplmethod> </aplresult>

Result: 0 ''

USER-PROPERTY

This option enables the APL+Win function to obtain the value of a specified user property.

xmlArg for USER-PROPERTY
<aplresult> <aplmethod>user-property</aplmethod> <apldata>userPropertyName</apldata> </aplresult>

Result: 0 userPropertyValue

USER-PROPERTIES

This option enables the APL+Win function to obtain the values of all user properties.

xmlArg for USER-PROPERTIES option
<aplresult> <aplmethod>user-properties</aplmethod> </aplresult>

Result: 0 propertiesArray

propertesArray is a two-column array with the 1st column the property name and the 2nd column the property value.

ERROR2APLCOM

This option enables the APL+Win function to cause AAS to return a specified error text as an http type 550 error message to the client.

xmlArg for ERROR2APLCOM
<aplresult> <aplmethod>error2aplcom</aplmethod> <apldata>errorMessage</apldata> </aplresult>

errorMessage is the desired error text string

Result: 0 ‘’

KILL2APLCOM

This option enables the APL+Win function to cause AAS to return a specified error text as an http type 560 error message to the client and close the instance of the APL+Win ActiveX engine which is executing the function.

xmlArg for KILL2APLCOM
<aplresult> <aplmethod>kill2aplcom</aplmethod> <apldata>errorMessage</apldata> </aplresult>

errorMessage is the desired error text string

Result: 0 ‘’

WHOAMI

This option enables the APL+Win function to determine which APLNext web server product and which web service has caused the function to be executed.

xmlArg for WHOAMI option

```
<aplresult>
<aplmethod>WHOAMI</aplmethod>
</aplresult>
```

Result: 0 ‘’

2nd element, if successful, is a two element vector of text strings containing the name of the APLNext web server product which received notification of the ‘onNotify’ event, e.g. ‘APLNextApplicationServer’ and the name of the APL+Win workspace in the AAS configuration which contains the APL+Win function which used the ‘Notify’ method’

BUSY

This option may be used to set the initial busy message to a desired text, ‘busyMessage’ when the APL+Win function is long-running.

xmlArg for BUSY option

```
<aplresult>
<aplmethod>update</aplmethod>
<aplbusy>busyMessage}</aplbusy>
</aplresult>
```

Result: 0 ‘’

UPDATE

This option may be used to set a subsequent busy message to a desired text string, ‘statusUpdate’ when the APL+Win function is long-running.

xmlArg for UPDATE option

```
<aplresult>
<aplmethod>update</aplmethod>
<aplstatus>statusUpdate</aplstatus>
</aplresult>
```

Result: 0 ‘’

START

This option enables the APL+Win function to cause AAS to start the instances of APL+Win ActiveX engine with the specified workspace according to its configuration in AAS.

xmlArg for START option

```
<aplresult>
<aplmethod>START</aplmethod>
<aplurl>wsid</aplurl>
</aplresult>
```

‘wsid’ is a virtual workspace name in the AAS configuration which is associated with the web service that is associated with the APL+Win function which used the ‘Notify’ method.

Result: 0 ''

STOP

This option enables the APL+Win to cause AAS to stop the instances of APL+Win ActiveX engine, if any, which are using the specified workspace.

xmlArg for STOP option
<aplresult> <aplmethod>STOP</aplmethod> <aplurl>wsid</aplurl> </aplresult>

'wsid' is the virtual workspace name in the AAS configuration which is associated with the web service that is associated with the APL+Win function which used the 'Notify' method.

Result: 0 ''

PAUSE

This option enables the APL+Win function to stop future client requests from being passed to APL+Win ActiveX instances which are using the specified workspace. This option does not prevent the specified instances of APL+Win ActiveX engine from making http requests or receiving requests from other APL+Win ActiveX instances in the AAS configuration.

xmlArg for PAUSE option
<aplresult> <aplmethod>PAUSE</aplmethod> <aplurl>wsid</aplurl> </aplresult>

'wsid' is the virtual workspace name in the AAS configuration which is associated with the web service that is associated with the APL+Win function which used the 'Notify' method.

Result: 0 ''

APLNext Desktop Server

APLNext Desktop Server (ADS) is the development and testing tool for APL+Win programmers to web-enable their application systems. ADS runs as a Windows application. Throughout this document only AAS is mentioned, however most features of AAS are also available in ADS. ADS includes one additional feature which makes it suitable for development and testing of an APL+Win application to be web-enabled.

Because AAS runs as a Windows service, for security reasons the server-side execution of APL+Win functions is not visible using the log-on credentials of the server machine. Microsoft correctly separates the web server profile from the log-on user profile. This means that while AAS is running the APL+Win

instances it creates may not be accessed by the APL+Win programmer, even if APL+Win developer version is registered on the server.

Since ADS runs as a Windows application, this security consideration does not apply as the log-on user has full access to the ADS Windows application. This means that the APL+Win programmer may view and modify the APL+Win instances created by ADS to facilitate development and testing (debugging).

To view and use the instances of APL+Win created by ADS for programming such as development or testing, register the APL+Win developer session on the target server machine, set the 'visible' property of the workspace to 1 and set the 'debug' property to 1. Generally it is easier to test the response of the server to a client request if the 'minpool' and 'maxpool' are both set to 1.