



▶ TCI/IP Support

▶ Writing a Web Server

Writing a Web Server

A sample Web server is provided in the *SERVER* namespace in the workspace *WWW.DWS*. It is capable of handling concurrent connections from several clients.

The main function is *SERVER.RUN* which is niladic and initialises the APL Web server using your default IP Address and port number 80.

To use the server, you must start a Web browser such as Netscape Navigator or Microsoft Internet Explorer. You may do this on another PC on your network or on your own PC. If so, you will probably find it most convenient to position your Dyalog Session Window and your browser window alongside one another.

To connect to the server, simply enter your user name (or your IP address) in the appropriate field in your browser (for example, `http://pete/`), and then press Enter.

When you press Enter, your browser will try to connect with a server whose IP address is your IP address and whose port number is 80; in short, the APL server. Upon connection, the following messages (but with different IP addresses) will appear in your Session window:

SERVER.RUN

Connected to 193.32.236.22

URL Requested:

Connected to 193.32.236.22

URL Requested: images/dyadic.gif

and the Dyalog home page will appear in your browser. This has in fact been supplied by your APL server. **NOTE IP Addresses must change to new IP Address**

The web server is implemented by the following functions:

<i><u>RUN</u></i>	user function to initiate an APL Web server
<i><u>ACCEPT</u></i>	callback which handles client connections
<i><u>RECEIVE</u></i>	callback which handles client commands
<i><u>ERROR</u></i>	callback which handles errors

Dyalog Ltd

South Barn
Minchens Court
Minchens Lane
Bramley
Hampshire
RG26 5BH
United Kingdom

Phone:

+ 44 (0) 1256 830 030

Fax:

+ 44 (0) 1256 830 031

e-mail:

sales@dyalog.com



▶ TCI/IP Support

▶ Writing a Web Server

The RUN function

```
[0]  RUN;CALLBACKS
[1]  DEX↑'TCPSocket'↓WN''
[2]  CALLBACKS←←('Event' 'TCPAccept' 'ACCEPT')
[3]  CALLBACKS,←←('Event' 'TCPRecv' 'RECEIVE')
[4]  CALLBACKS,←←('Event' 'TCPErrror' 'ERROR')
[5]  COUNT←0
[6]  'SO'↓WC'TCPSocket' '' 80,CALLBACKS
```

RUN[1] expunges all TCPSocket objects that may be already defined. This is intended only to clear up after a potential error.

RUN[2 - 5] set up a variable *CALLBACKS* which associates various functions with various events.

RUN[6] initialises a variable *COUNT* which will be incremented and used to name new TCPSocket objects as each client connects. *COUNT* is global within the *SERVER* namespace.

RUN[7] creates the first TCPSocket server using your default IP address and port number 80.

Once the server has been initiated, the next stage of the process is that a client makes a connection. This is handled by the *ACCEPT* callback function.

The ACCEPT callback function

```
[0]  ACCEPT MSG;SOCK;EVENTS
[1]  COUNT←COUNT+1
[2]  SOCK←('SocketNumber' (3=MSG))
[3]  EVENTS←('Event' (←MSG)↓WG'Event')
[4]  ('S',*COUNT)↓WC'TCPSocket'SOCK EVENTS
[5]  DCS=MSG
[6]  'Connected to ',(↓WG'RemoteAddr')
[7]  BUFFER←DAVE[4 3]
```

The *ACCEPT* function is invoked when the TCPAccept event occurs. This happens when a client connects to the server.

Its argument *MSG* , supplied by APL, is a 3-element vector containing

Dyalog Ltd

South Barn
Minchens Court
Minchens Lane
Bramley
Hampshire
RG26 5BH
United Kingdom

Phone:

+ 44 (0) 1256 830 030

Fax:

+ 44 (0) 1256 830 031

e-mail:

sales@dyalog.com



▶ TCI/IP Support

▶ Writing a Web Server

MSG[1] The name of the TCPSocket object

MSG[2] The name of the event ('*TCPAccept*')

MSG[3] The socket handle for the original listening socket

ACCEPT[1] increments the *COUNT* variable. This variable is global to the *SERVER* namespace and was initialised by the *RUN* function.

ACCEPT[3] makes a new TCPSocket object called *Sxx*, where *xx* is the value of *COUNT*. By specifying the socket handle of the original listening socket as the value of the SocketNumber property for the new object, this effectively clones the listening socket.

ACCEPT[4] changes to the namespace of the TCPSocket object, that is now connected to a client.

ACCEPT[5] displays the message *Connected to xxx.xxx.xxx.xxx*, the IP address of the client, which is obtained from the value of the RemoteAddr property.

ACCEPT[6] initialises a variable *BUFFER* to `AV[4 3]` (CR,LF). This variable is global to the *SERVER* namespace and is used by the *RECEIVE* callback function to accumulate the command that is transmitted by the client. This happens next.

Dyalog Ltd

South Barn
Minchens Court
Minchens Lane
Bramley
Hampshire
RG26 5BH
United Kingdom

Phone:

+ 44 (0) 1256 830 030

Fax:

+ 44 (0) 1256 830 031

e-mail:

sales@dyalog.com



▶ [TCI/IP Support](#)

▶ [Writing a Web Server](#)

The RECEIVE callback function

```
[0] RECEIVE MSG;CMD;OLD;URL;FILE;DATA
[1] OLD←DCS⇒MSG
[2] BUFFER,←3⇒MSG
[3] :If DAV[4 3 4 3]≠~4↑BUFFER A Have we got everything
[4]     :Return
[5]     :EndIf
[6]
[7] CMD←2↓"(DAV[4 3]∈BUFFER)←BUFFER
[8] CMD←⇒CMD A Ignore everything except client request
[9] DCS OLD
[10] :If 'GET /'≡5↑CMD
[11]     URL←5↓CMD
[12]     URL←(↑1+URL\.'')↑URL
[13]     Q←'URL Requested: ',URL
[14]     :If 0=↑URL ∅ URL←'index.htm' ∅ :EndIf
[15]     URL←(-'.html'≡↑5↑URL)↓URL
[16]     FILE←2 DNQ"." 'GetEnvironment' 'Dyalog'
[17]     FILE,←'\samples\tcpip\homepage\' ,URL
[18]     DATA←GETFILE FILE
[19]     :If 0<↑DATA
[20]         2 DNQ(=MSG)'TCPsend'DATA
[21]     :EndIf
[22] :EndIf
[23] DEX⇒MSG
```

The *RECEIVE* function is invoked whenever a TCPRecv event occurs. This happens when a data packet is received from a client. Its argument *MSG*, supplied by APL, is a 3-element vector containing::

MSG[1] The name of the TCPSocket object

MSG[2] The name of the event ('*TCPReceive*')

MSG[3] The data

RECEIVE[1] changes to the namespace of the TCPSocket object. The name of the current namespace is stored in the local variable *OLD*.

RECEIVE[2] concatenates the newly received data packet to the *BUFFER* variable that is encapsulated in the TCPSocket object and was initialised by the *ACCEPT* function.

Dyalog Ltd

South Barn
Minchens Court
Minchens Lane
Bramley
Hampshire
RG26 5BH
United Kingdom

Phone:

+ 44 (0) 1256 830 030

Fax:

+ 44 (0) 1256 830 031

e-mail:

sales@dyalog.com



▶ TCI/IP Support

▶ Writing a Web Server

Dyalog Ltd

South Barn
Minchens Court
Minchens Lane
Bramley
Hampshire
RG26 5BH
United Kingdom

Phone:

+ 44 (0) 1256 830 030

Fax:

+ 44 (0) 1256 830 031

e-mail:

sales@dyalog.com

RECEIVE
[3-5]

tests whether or not all of the command sent by the client has been received. This is true only if the last 4 characters of *BUFFER* are CR,LF,CR,LF. If there is more data to come, *RECEIVE* exits; otherwise it goes on to process the command.

RECEIVE
[7-8]

splits the command into sub-strings and then discards all but the first one.

RECEIVE[9]

changes back into the *SERVER* namespace

RECEIVE
[10-11]

passes the client request for a URL. For the sake of simplicity, the request is assumed to begin with the string '*GET /*'. If not, the request is ignored.

RECEIVE
[12]

removes all trailing information that might be supplied by the browser after the URL.

RECEIVE
[14]

checks for a request for an empty URL (which equates to the home page). If so, it substitutes *index.htm* which is the name of the file containing the home page.

RECEIVE
[15]

changes the file extension of the URL from *.html* to *.htm* if required.

RECEIVE
[16]

sets the value of local variable *FILE* to the name of the directory in which Dyalog is installed.

RECEIVE
[17]

appends the pathname of the subdirectory *\samples\tcpip\homepage* and the name of the URL. *FILE* now contains the full pathname of the requested web page file.

RECEIVE
[18]

uses the utility function *GETFILE* to read the contents of the file into the local variable *DATA*.

RECEIVE
[19]

checks that the result of *GETFILE* was not empty and if so, does nothing. This would be the case if the file did not exist.

RECEIVE
[20]

uses TCPSend to transmit the contents of the file to the browser.

RECEIVE
[23]

expunges the TCPSocket object. This is a fundamental part of the HTTP protocol because when the client socket subsequently gets closed, it knows that all of the data transmitted by the server has been received.