The Trainer's Friend
INCORPORATED

Setting Up the IBM HTTP Server

- **A "near-cookbook" to get you going**

by Steve Comstock

The Trainer's Friend, Inc.
http://www.trainersfriend.com
303-393-8716
steve@trainersfriend.com

v2.1

Acknowledgements

The following people were instrumental in helping me be successful in this project:

Hunter Cobb, The Trainer's Friend, Inc.
David Day, Duke Software
Bob Lester, Oppenheimer Funds
Jack Shudel, University of Florida
Alan Star, CaLPERS
Jim Thomas

Any errors found here are strictly my own, and I appreciate any comments, suggestions, or corrections.

Table of Contents

Introduction

This paper was designed to help you get set up quickly with V5R3M0 of the IBM HTTP Server (IHS). It is not exhaustive nor definitive, it is simply a process that worked for me.

This server comes free with z/OS; it has been around a long time and is not likely to be updated. Although it is older technology than the ported Apache server that you can order for no charge, I find this server meets my needs better than the Apache server.

The definitive source for this process is the IBM publication "HTTP Server Planning, Installing, and Using" found at

http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/BOOKS/imwziu18/CCONTENTS

This paper is a quick path, ignoring all the complexities, focusing on just enough to get the server going and then some basics of configuration. I am not a systems programmer, so you may find or know of better ways to do this. I'm just saying the process described here should work well enough to get you going.

Our objective is to have the IHS serving pages from a single z/OS image. We generally ignore issues of sharing files and cross-sysplex communication, etc. Our focus on security is minimal. Once you have a working server you can go back and make adjustments as needed for your installation.

Introduction, p.2.

Basic assumptions here:

* you are using RACF (although other security packages work fine, and you simply need to convert the RACF statements to the corresponding statements in your security package)

* you are using SDSF (but any package that allows you to display JES queues and issue system commands will work fine)

* z/OS UNIX is already running in full-function mode

  (if you are using TCP/IP, you are running in full-function mode)

* You have necessary authority to issue RACF commands (or commands for whatever security product you use) and to become superuser under omvs

* You know (or can find out) the IP address for your z/OS system, and / or you have a domain name that maps to your z/OS system

There is some possibilty that some of the work to set up the HTTP server has already been done. Maybe you or some other systems programmer started out setting this up and then could not get around to completing it. Maybe the system was delivered to you with part of the setup done (for example, if you run off the IBM Dallas shared system). So we include in most steps a check first to see if a step is really necessary.

Overview

There are many pieces that have to fit together to get the server working; you will need to ...

- ♦ define a user group for the web server and web administrator
- ♦ define user IDs for the web server and web adminstrator (RACF or other security product)
- ♦ add an omvs segment to the RACF entry for each user who will be using the system
- ♦ allocate some z/FS files
- ♦ connect files into the UNIX file hierarchy
- ♦ modify or add PARMLIB members
- ♦ modify TCP/IP configuration
- ♦ set up JCL to run the server
- ♦ copy configuration files
- ♦ configure the server
- ♦ start up the server and confirm connection

Defining New IDs

In order to run the server, we will need some user IDs (identities) for the server to use, and we need to set up a group for these users to belong to.

This work is accomplished through RACF commands (or equivalent commands for whatever security product you are using).

There are at least four ways to issue RACF commands

- ♦ Issue RACF commands directly from ISPF 6 or even TSO READY prompt
- ♦ Embed the commands in a REXX exec or CLIST
- ♦ Embed the commands in a batch job
- ♦ Use the ISPF RACF panels

It seems that when there's only one or two commands to be issued that RACF adminstrators tend to issue the commands directly from ISPF 6 but if there are multiple commands they tend to be issued from a batch job. We will take that approach here.

Define the IMWEB Group

We need to have a containing group for our web server IDs, and IMWEB is the default / preferred name. Creating this group allows us to handle a collection of related user id's under a single umbrella

First, to check if this has already been done, issue this command:

```
==> lg imweb
```

If you get some lines back beginning with:

```
INFORMATION FOR GROUP IMWEB
```

then you're good to go; but if not, issue this command:

```
==> addgroup imweb omvs(gid(205))
```

- ♦ "GID" is the Group ID number; each group has a unique identifying number, an integer between 0 and 2147483647

- ♦ The GID of 205 for IMWEB is expected (it is hard coded in some supporting scripts we'll encounter later)

  - ✗ If you change the GID here you will need to change the scripts

Add WEBADM and WEBSRV IDs

For access to TSO, you need a RACF ID, what we usually call a "TSO ID" or "logon id".

Similarly, for access to z/OS UNIX you need a UNIX ID (a name; by convention this is your TSO ID in lower case), and a UID (an integer in the range 0-2147483647).

UID '0' is superuser (can issue any command against any resource)

Best practices:

1) assign a unique UID to each user with OMVS access

2) minimize the number of users with UID 0.

The WEBADM ID is for the person responsible for administering the web server's configuration, and for monitoring problems.

WEBSRV is the ID that the HTTP server itself will run under; when the HTTP server is started up, this is the id that will be used. Note that the doc shows WEBSRV using UID 0, so we will also use that; however, there are some notes in the central publication that demonstrate how to use a non-0 UID for WEBSRV.

There is a way for a non-0 UID to switch to superuser mode when needed, and we discuss that shortly (see p. 15).

The UNIX ID is also called the 'OMVS ID'. Note that you can have a TSO ID without an OMVS ID, and an OMVS id without a TSO ID. The WEBADM and WEBSRV OMVS IDs, for example, should not be defined with any TSO access. (Note also these IDs are all uppercase, so a little out of the ordinary.)

Add WEBADM and WEBSRV IDs, continued

First, to check if these IDs already exist issue

```
   ==> lu (webadm, websrv)
```

If the response includes

```
ICH30001I UNABLE TO LOCATE USER     ENTRY WEBADM
or
ICH30001I UNABLE TO LOCATE USER     ENTRY WEBSRV
```

... you will need to define these IDs, using these commands:

```
   ADDUSER WEBADM DFLTGRP(IMWEB) OMVS(UID(206) -
        HOME('/usr/lpp/internet') PROGRAM('/bin/sh'))

   ADDUSER WEBSRV DFLTGRP(IMWEB) OMVS(UID(0)    -
        HOME('/usr/lpp/internet') PROGRAM('/bin/sh'))
```

- ♦ Note: z/OS UNIX is generally case sensitive
  - ✗ You can enter RACF commands in upper-, lower-, or mixed-case, but z/OS UNIX file names need to follow the exact capitalization for names and UNIX commands must be lower-case (this is why the names above are in quotes)
- ♦ See the next page for embedding these commands in a job

Add WEBADM and WEBSRV IDs, continued

In the context of a job you might code:

```
//ADDIDS   JOB    ...
//S0       EXEC PGM=IKJEFT01,DYNAMNBR=75,TIME=100,REGION=6M
//SYSPRINT  DD  SYSOUT=*
//SYSTSPRT  DD  SYSOUT=*
//SYSTERM   DD  DUMMY
//SYSUADS   DD  DSN=SYS1.UADS,DISP=SHR
//SYSLBC    DD  DSN=SYS1.BRODCAST,DISP=SHR
//SYSTSIN   DD  *
ADDUSER WEBADM DFLTGRP(IMWEB) OMVS(UID(206) -
     HOME('/usr/lpp/internet') PROGRAM('/bin/sh'))
ADDUSER WEBSRV DFLTGRP(IMWEB) OMVS(UID(0)    -
     HOME('/usr/lpp/internet') PROGRAM('/bin/sh'))
```

- ♦ When you create a job like this, be sure you are editing with CAPS OFF mode so the lower case file names don't get upper-cased

  - ✗ But then, of course, be sure your JCL statements are all upper-case!

- ♦ The UID of 206 for WEBADM and 0 for WEBSRV are the expected values

Facility Class Concerns

If you have defined the BPX.DAEMON profile in the FACILITY class, you need to ensure that WEBSRV has READ access to it.

If you have defined the BPX.SERVER profile in the FACILITY class, WEBSRV must have UPDATE acces to it.

If these profiles are not defined, you also need to define them.

To see if these profiles are defined, issue this command:

```
   ==> rl facility (bpx.daemon, bpx.server)
```

if you get back messages like

```
ICH13003I BPX.DAEMON NOT FOUND
ICH13003I BPX.SERVER NOT FOUND
```

... then you need to issue this string of commands (probably in a batch job):

```
RDEFINE FACILITY BPX.DAEMON UACC(NONE) NOTIFY(WEBSRV)
RDEFINE FACILITY BPX.SERVER UACC(NONE) NOTIFY(WEBSRV)
PERMIT BPX.DAEMON CLASS(FACILITY) ID(WEBSRV) ACCESS(READ)
PERMIT BPX.SERVER CLASS(FACILITY) ID(WEBSRV) ACCESS(UPDATE)
SETROPTS RACLIST(FACILITY) REFRESH
```

- On the other hand, if you found that the profiles were already defined, you can skip the first two commands above, but still issue the last three

Adding OMVS Segments For Users

For each existing TSO user who is to get access to z/OS UNIX you will need what is called an "OMVS segment" in their RACF (or other security product) definition. First, of course, we check to see if this has already been done. From ISPF 6 try this command:

```
   ==> lu userid omvs
```

... if this user has an omvs segment already, you will see a set of lines something like this at the end of the display:

```
        OMVS INFORMATION
        ----------------
        UID= 00000000xx
        HOME= /u/userid
        PROGRAM= /bin/sh
        CPUTIMEMAX= NONE
        ASSIZEMAX= NONE
        FILEPROCMAX= NONE
        PROCUSERMAX= NONE
        THREADSMAX= NONE
        MMAPAREAMAX= NONE
```

... if there is no omvs segment, you'll see this at the end:

```
        NO OMVS INFORMATION
```

Adding OMVS Segments For Users, continued

For each user you intend to give access to z/OS UNIX who does not already have it, you need to modify their ID using the ALTUSER command, something like:

```
ALU userid OMVS(HOME('/u/userid') UID(nn) PROGRAM('/bin/sh'))
```

Notes

- The second *userid* should be in lowercase; however, whatever case you supply will be preserved
- /bin/sh must be in lowercase
  - ✗ Be sure you are editing with a profile settng of CAPS OFF
- The *nn* must be an integer between 0 and 2147483647 that is unique for this user
  - ✗ It's probably a good idea to have some kind of convention in your shop when assigning UID values
- Some of our work will require becoming super user; the following page discusses a way to gain the ability to become a superuser without having a UID of 0

Again, if you have a lot of user IDs to update, it's probably best to build a batch job and use the power of the editor to help you generate all the commands.

The BPX.SUPERUSER Facility Class

Very few z/OS UNIX users or programs really need to run as superuser. In fact, having the superuser designation can be a pretty dangerous thing, either by design or by accident.

Happily there is a way to give users a non-zero UID and still allow them to switch into (and out of) superuser mode for just the period of time they need to accomplish a specific task that requires they be superuser.

The trick is to grant READ access to the BPX.SUPERUSER Facility class.

The RACF PERMIT command (abbreviated PE) is the way to go:

```
   PE BPX.SUPERUSER CLASS(FACILITY) ACCESS(READ) ID(userid)
```

gives permission to issue the 'su' (superuser) command when running z/OS UNIX commands; this changes the authority of the issuer to have superuser abilities without a UID of 0. Once in superuser mode, the 'exit' command takes you back to your standard non-superuser status.

Creating zFS Data Sets

Each user will need their own zFS ("zSeries File System") as a container for their z/OS UNIX files. So first, we will allocate and format a zFS data set for each TSO user who will be using z/OS UNIX files. Also, we are planning to put all our web server configuration files into a zFS data set.

We recommend you set up a batch job for each separate zFS. Something like this:

```
//ZFSBLD    JOB ...
//*   This job defines and formats a ZFS data set to hold
//*   user SCOMSTO's z/OS UNIX files
//*
//* IOEAGFMT is documented in SC24-5989
//*     Distributed File Service ZFS Administration
//*
//DEFINE    EXEC    PGM=IDCAMS
//SYSPRINT DD      SYSOUT=*
//SYSIN     DD      *
      DEFINE CLUSTER (NAME(SCOMSTO.ZFS) -
              VOLUMES(volser) -
              LINEAR CYL(600 40) SHAREOPTIONS(3))
//CREATE    EXEC    PGM=IOEAGFMT,REGION=0M,
// PARM=('-aggregate SCOMSTO.ZFS -compat')
//SYSPRINT DD      SYSOUT=*
//STDOUT    DD      SYSOUT=*
//STDERR    DD      SYSOUT=*
//CEEDUMP   DD      SYSOUT=*
```

- SCOMSTO is my actual id on the system I work on, so I've left it in to make some of the examples more concrete

- More notes on the following page:

Creating zFS Files, continued

Notes

- ♦ Notice that a zFS is actually a VSAM LINEAR DATA SET
- ♦ The size on this file is pretty large, but this user was expected to use lots of space
  - ✗ If you expect a zFS may need to grow, specify some non-zero secondary space amount
  - ✗ Normally the maximum is a single volume, but the doc "Distributed File Service ZFS Administration" describes how to create a multi-volume zFS
- ♦ We use a naming convention of *userid*.ZFS for our ZFS files, but you can use just about any valid MVS data set name you'd like
  - ✗ For example, you might have ACS routines that direct you to a specific [pool of] volume[s], based on the cluster name
- ♦ We set up a similar job for each user we expect to give access to z/OS UNIX; note there are ways to automate this process
- ♦ We don't need to allocate a zFS for WEBADM nor WEBSRV since their HOME location is /usr/lpp/internet, and the necessary files come with z/OS

Creating zFS Files, continued

Note that zFS files are somewhat new. Older versions of z/OS may have to use HFS (Hierachal File System) file containers. Here's an example of allocating an HFS file container:

```
ALLOC DA('SMSTO.HFS') DSNTYPE(HFS) SPACE(10 5) CYL DIR(1)
   NEW CAT
```

Note that both HFS and zFS files can be allocated using ISPF 3.2 (but the zFS will not be formatted without running the IOEAGFMT utility).

Also note that z/OS 1.13 introduced some changes in how zFS file systems are handled, and the reader is directed to the z/OS 1.13 UNIX System Services Planning manual for information on that. This is primarily sysplex related.

So at this point, we have set up all the first users of z/OS UNIX with their own OMVS segment and their own zFS (or HFS) file containers. We need a little bit more, but first we need to take a short digression to introduce some terms and vocabulary.

z/OS UNIX Files

The zFS (and HFS) we've discussed are containers for your actual z/OS UNIX files. It might help to keep the distinction of the zFS and HFS data sets (an MVS term) as containers for your UNIX files.

UNIX files are either directories or files. Actually, a directory is just a file that contains other files and / or directories. The top directory in the container is the root, and we identify it using a slash (/). Below the root we can have subdirectories and files.

A file name can be 1-256 characters (case-sensitive; some special characters are permitted, but life is simple if you only allow alpha-numeric (a-z, A-Z, 0-9) and a few special characters, say just underscore and dash (_ and -). This way you avoid a lot of work in issuing various UNIX commands.

Each directory may contain regular files and sub-directories. A regular file is located by specifying the chain of directories under which the file is found. For example:

```
/u/scomsto/public_html/Welcome. html
```

The above locates the file 'Welcome.html' as being in the sub-directory **public_html** which is a sub-directory of **scomsto** under the directory of **u** under the root. Everything up to the filename itself is called the path. Path names, in z/OS UNIX, can be up to 1023 characters long.

This hierarchy of directories and files provides a nice way to organize and group files in a logical fashion.

z/OS UNIX Files, 2

You want to physically separate hierarchies of files in their own containers. For example, each user has their own zFS, with the root in the container being their UNIX ID. Below that they may have any set of directories and files they wish. Then, for example, backing up their files is simply a matter of copying their containing data set.

At the very top of the z/OS UNIX file hierarchy is the system root (/). Below that are a set of common directories, typically including:

bin - for system commands and other executables
dev - for device assignments
etc - for system support files
lib - for data libraries
tmp - for temporary files
u - where we will put (mount) individual user's file systems
usr - for software products
var - for variable definitions

Most of these first level directories come with z/OS so you needn't worry about allocating containers and the like for them. But we would like to add a new directory of our own just below root level:

web - a place to gather all the configuration and log files for various HTTP servers

In our system, for example, below web we have a file collection for the IBM HTTP Server, but also one for the ported Apache, as well as one for a version of Apache that I ported myself. So this is a convenient place to plan on putting all web-server support files.

z/OS UNIX Files, 3

To start, in our installation we decided to put our configuration files for our HTTP web server under /web

This means we'll need to allocate a zFS, something like this:

```
//ZFSBLD    JOB ...
//*   This job defines and formats a ZFS data set for
//*   our web subdirectory
//*
//DEFINE    EXEC    PGM=IDCAMS
//SYSPRINT DD      SYSOUT=*
//SYSIN     DD      *
      DEFINE CLUSTER (NAME(WEB.ZFS) -
              VOLUMES(volser) -
              LINEAR CYL(2 4) SHAREOPTIONS(3))
//CREATE    EXEC    PGM=IOEAGFMT,REGION=0M,
// PARM=('-aggregate WEB.ZFS -compat')
//SYSPRINT DD      SYSOUT=*
//STDOUT    DD      SYSOUT=*
//STDERR    DD      SYSOUT=*
//CEEDUMP   DD      SYSOUT=*
```

Notes

- This only needs a small amount of space, since it will just contain /web - other subdirectories can be added later

- However, the use of WEB for the high level qualifier may cause some problems ...

z/OS UNIX Files, 4

When you allocate an MVS file, you want the file cataloged. But you don't want it cataloged in the system master catalog (unless it is a system support data set).

Instead, when you catalog a user data set, you want it cataloged in a user catalog. For allocation routines to be able to find data sets that are cataloged in user catalogs, you need to define aliases in the master catalog that point to the right user catalog. This is done by mapping the high level qualifier to the right user catalog using 'DEFINE ALIAS' IDCAMS commands.

But if you have a lot of different user high level names, you can clutter up the master catalog with lots of aliases.

Solution: create just a few high level qualifiers and group similar files under those high level qualifiers. For example, we use MANAGE as a high level qualifier and so we might want to change the previous job to something like this:

```
//ZFSBLD    JOB ...
//*   This job defines and formats a ZFS data set
//*   for our web subdirectory
//*
//DEFINE    EXEC    PGM=IDCAMS
//SYSPRINT DD       SYSOUT=*
//SYSIN     DD       *
     DEFINE CLUSTER (NAME(MANAGE.WEB.ZFS) -
            VOLUMES(volser) -
            LINEAR CYL(2 4) SHAREOPTIONS(3))
//CREATE    EXEC    PGM=IOEAGFMT,REGION=0M,
// PARM=('-aggregate MANAGE.WEB.ZFS -compat')
//SYSPRINT DD       SYSOUT=*
//STDOUT    DD       SYSOUT=*
//STDERR    DD       SYSOUT=*
//CEEDUMP   DD       SYSOUT=*
```

z/OS UNIX Files, 5

Finally, SMS may enter the equation here. zFS data sets do not need to be SMS managed, but they may be, and there are some advantages to set up zFS data sets as SMS managed.

So, you need to make some decisions about storage management at this point:

- Will your zFS data sets be SMS managed?
- What volumes (or storage group) should you use?
- What high level qualifier(s) will you use (note that for zFS data sets for each TSO user, it's best to use the userid as the high level qualifier, since userids will almost certainly have aliases already set up)

Again, with our stated goal of keeping things simple, we chose a non-SMS-managed volume (volume serial of VPWRKD) that we had read / write access to and, initially at least, put all our non-system zFS's on this volume, and we will use a generic high level qualifier of MANAGE for non-user-specific data sets.

We cannot run the formatting utility to format multiple zFS's in a single step since the request is sent as a PARM string. So we will run a job with two steps for each zFS we will allocate and format. This can be a series of jobs or one job with lots of steps.

On the following pages we demonstrate creating the data set we'll need for our web related files plus the zFS data set for one user ...

z/OS UNIX Files, 6

```
//ZFSBLD    JOB ...
//*  This job defines and formats a ZFS data set for
//*  our web subdirectory and for SCOMSTO
//*
//DEFINE1  EXEC  PGM=IDCAMS
//SYSPRINT  DD   SYSOUT=*
//SYSIN     DD   *
      DEFINE CLUSTER (NAME(MANAGE.WEB.ZFS) -
              VOLUMES(VPWRKD) -
              LINEAR CYL(2 4) SHAREOPTIONS(3))
//FORMAT1  EXEC   PGM=IOEAGFMT,REGION=0M,
// PARM=('-aggregate MANAGE.WEB.ZFS -compat')
//SYSPRINT DD      SYSOUT=*
//STDOUT   DD      SYSOUT=*
//STDERR   DD      SYSOUT=*
//CEEDUMP  DD      SYSOUT=*
//*
//DEFINE2   EXEC   PGM=IDCAMS
//SYSPRINT   DD    SYSOUT=*
//SYSIN      DD    *
      DEFINE CLUSTER (NAME(SCOMSTO.ZFS) -
              VOLUMES(VPWRKD) -
              LINEAR CYL(600 40) SHAREOPTIONS(3))
//FORMAT2   EXEC   PGM=IOEAGFMT,REGION=0M,
// PARM=('-aggregate SCOMSTO.ZFS -compat')
//SYSPRINT  DD      SYSOUT=*
//STDOUT    DD      SYSOUT=*
//STDERR    DD      SYSOUT=*
//CEEDUMP   DD      SYSOUT=*
```

So we now end up with 2 allocated and formatted zFS data sets: MANAGE.WEB.ZFS and SCOMSTO.ZFS. Next, how to connect these to the system root ...

Connecting Files Into the UNIX File Hierarchy

Recall that the UNIX file hierarchy starts with the root and just a handful of sub-directories; in a simplified diagram:

```
                    /                  (root)
   ---------------------------------------
     |        |          |  .    .    .
   etc        u         usr
```

Initially, we want to add our 'web' directory at this level.

First, of course, we need to check if this directory is already defined (not likely, but possible). Simplest approach is to use ISPF 3.17 and fill in the first input field like:

```
  Menu   RefList   RefMode   Utilit
--------------------------------
                          z/OS  UNIX
Option ===>_________________________

   blank Display directory list

   Pathname . . . /web____________
```

If the directory does not exist, the short message area will contain '**Directory not found**'.

Older systems don't have ISPF 3.17; in this case go to ISPF 6 and issue: **oshell cd /web** and if /web does not exist you'll get an error message

Connecting Files Into the UNIX File Hierarchy, 2

To create the directory 'web' at the level below the root use the TSO mkdir command. From ISPF 6 issue:

```
      ===> mkdir '/web'
```

Note: you must have sufficient authority to have permission to add a file or directory at the root level.

The above mkdir command creates a directory named 'web' off the root directory. This directory is visible to everyone using z/OS UNIX and will persist until it is explicitly removed.

Once you have a directory, you can attach your zFS (or HFS) container data set at that directory point. This process is called 'mounting' and the directory is called the 'mount point'. And, not surprisingly, you accomplish this using the 'mount' command! From ISPF 6 you can issue this:

```
===> mount filesystem('MANAGE.WEB.ZFS')
        mountpoint('/web') type(zfs) mode(rdwr)
```

(this will actually fit on a single line)

So you can think of /web as the place where z/OS UNIX files stored in MANAGE.WEB.ZFS will be found.

Connecting Files Into the UNIX File Hierarchy, 3

Similar remarks apply to our user's zFS. We need to make a directory under '/u', so:

```
    ===> mkdir '/u/scomsto'
```

Then to mount our user zFS under '/u' we might issue:

```
===> mount filesystem('SCOMSTO.ZFS')
       mountpoint('/u/scomsto') type(zfs) mode(rdwr)
```

And our z/OS UNIX file hierarchy now looks, in part, like:

```
                     /                    (root)
    ----------------------------------------
      |        |          |  ...     |
    etc        u         usr ...   web
               |
              scomsto
```

Note: in my work, we set up a separate zFS for each user. This is not necessary: many users may have their directories in a single zFS. Also, we put server stuff in a non-standard place. That is: configuration files are typically stored under /usr/lpp/internet/etc, while logs, pubs, reports, and so on are typically stored under /usr/lpp/internet/server_root.

Since we often run several servers, we actually use this structure for holding configuration files as well as logs, pubs, and reports:

/web/Apache - for the IBM ported Apache
/web/httpd1 - for the IBM HTTP Server (IHS) V5R3M0
/web/zApache - for my ported Apache

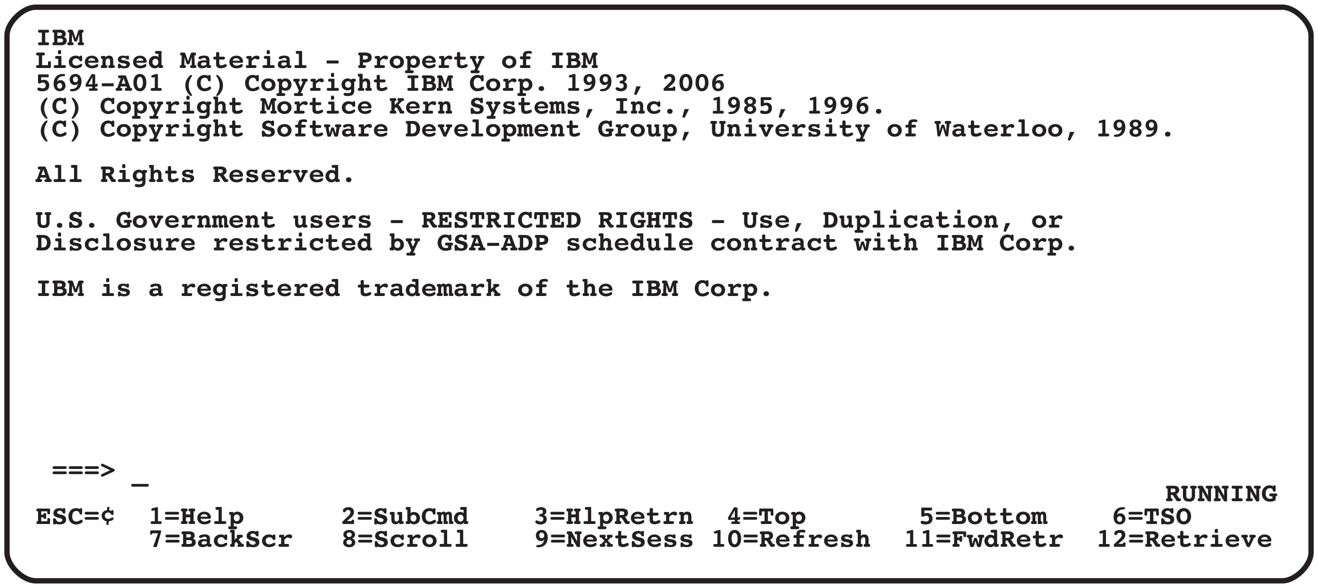
Confirming OMVS Access

At this point, you can test if your user ID(s) can get into OMVS!

From ISPF 6, issue this command:

```
===>  omvs
```

You should see a screen something like this:



```
IBM
Licensed Material - Property of IBM
5694-A01 (C) Copyright IBM Corp. 1993, 2006
(C) Copyright Mortice Kern Systems, Inc., 1985, 1996.
(C) Copyright Software Development Group, University of Waterloo, 1989.

All Rights Reserved.

U.S. Government users - RESTRICTED RIGHTS - Use, Duplication, or
Disclosure restricted by GSA-ADP schedule contract with IBM Corp.

IBM is a registered trademark of the IBM Corp.






 ===> _
                                                                      RUNNING
ESC=¢  1=Help      2=SubCmd    3=HlpRetrn  4=Top       5=Bottom    6=TSO
       7=BackScr   8=Scroll    9=NextSess 10=Refresh  11=FwdRetr  12=Retrieve
```

We're not where we want to be yet, but this is a positive sign we are on the right track.

To leave this screen and return to ISPF, enter the command: **exit** (it will take 2 <Enter>s to get back).

Now, back to work.

Modifying IPL Parms

The mount commands discussed on the previous pages will not be remembered across IPLs. To make sure the mounts are re-established with each IPL, you can add mount parameters to one or more of the BPXPRM***xx*** members referenced at IPL time.

How to find which members are currently being used? One way:

1. From an SDSF (or equivalent) LOG command issue:

```
f bpxprm first
```

then you should see a series of messages, like:

```
IEE252I MEMBER BPXPRMVN FOUND IN VENDOR.PARMLIB
IEE252I MEMBER BPXPRM66 FOUND IN SVTSC.PARMLIB
IEE252I MEMBER BPXPRMRD FOUND IN SVTSC.PARMLIB
IEE252I MEMBER BPXPRMFM FOUND IN SVTSC.PARMLIB
IEE252I MEMBER BPXPRM61 FOUND IN SVTSC.PARMLIB
IEE252I MEMBER BPXPRM70 FOUND IN SVTSC.PARMLIB
IEE252I MEMBER BPXPRMIA FOUND IN SVTSC.PARMLIB
IEE252I MEMBER BPXPRMI1 FOUND IN SVTSC.PARMLIB
IEE252I MEMBER BPXPRMDB FOUND IN VENDOR.PARMLIB
IEE252I MEMBER BPXPRMSV FOUND IN VENDOR.PARMLIB
IEE252I MEMBER BPXPRMOM FOUND IN LVL0.PARMLIB
IEE252I MEMBER BPXPRMWM FOUND IN SVTSC.PARMLIB
```

This tells you the members and their libraries. Choose an appropriate library and then one of the members from that library.

In our case, for example, VENDOR.PARMLIB is for members we want to override members of the same name in other parmlibs.

Modifying IPL Parms, 2

Here, the member to use depends on any installation standards. For example, we use one member for DB2-related parms, another for other program products (compilers, and so on) and another for user-related mounts. You might, thus, add these commands to different PARMLIB members (maybe even in different PARMLIBs):

```
.
.
.

/*******************************************/
/* ADDITIONAL CUSTOMER MOUNTS              */
/*******************************************/

MOUNT     FILESYSTEM('SCOMSTO.ZFS')
           TYPE(ZFS)
           MODE(RDWR)
           MOUNTPOINT('/u/scomsto')
.
.
.

MOUNT     FILESYSTEM('MANAGE.WEB.ZFS')
           TYPE(ZFS)
           MODE(RDWR)
           MOUNTPOINT('/web')
```

Modifying TCP/IP parms

To modify your TCP/IP parms to support your web server, you have to find out where these parms are found. Here is one way to find out where the parms are:

1. From SDSF, issue the DA (Display Active) command and find the task named TCPIP (if there is no such task, you must find out from someone in your shop what name is used for the TCPIP process)

2. Select this task ('S' next to the task name, in the NP column) and look at the JCL listing there for a DD name of PROFILE

   the DSN will tell you where to find your TCP/IP profile settings

3. The PORT entries will tell you if you have set up an entry for your HTTP server. Usually port 80 is used and one would expect to see something like:

```
80 TCP HTTPD1                          ; HTTP Server
```

   If you don't see this, add it (talk with your network staff first; see if there is some other port you should use)

   Note that 'HTTPD1' is the name we expect to use for the proc for starting up the server

4. While you're looking at your TCP/IP parms, look for the HOME entry, which should have your IP address! Jot this down for later.

Modifying TCP/IP parms, 2

5. If you changed the TCP/IP profile, the change will be picked up at the next IPL; to have the change take effect immediately, issue this console command:

```
vary tcpip,,obeyfile,tcpip_profile
```

- Where 'tcpip' is the name of the TCP/IP proc, and 'tcpip_profile' is the fully qualified data set name of the TCP/IP profile data set (this can be a sequential file or a PDS with membername), specified without quotes

- Instead of issuing this from the console, you can issue this from an SDSF (or equivalent) command line, assuming you have the authority to do so; maybe:

```
/vary tcpip,,obeyfile,tcpip_profile
```

Hint: if there is not enought room on the SDSF command line, enter just a slash and press <Enter>. This will give you a pop-up panel large enough to enter the whole command (without the leading slash).

Set up JCL to run the Server

In SYS1.SAMPLIB, member IMWEBSRV is a model proc for starting the server. But we ended up with a pretty different version that works well for us, so here's the JCL we use:

```
//HTTPD1 PROC   P1='-B',
// P2='-r /web/httpd.conf',
// P3='ENVAR("_CEE_ENVFILE=/web/httpd.envvars")/ -vv'
//WEBSRV1   EXEC PGM=IMWHTTPD,REGION=0K,TIME=NOLIMIT,
//    PARM=('&P3 &P2 &P1')
//SYSIN      DD DUMMY
//OUTDSC  OUTPUT DEST=HOLD
//SYSPRINT  DD SYSOUT=*,OUTPUT=(*.OUTDSC)
//SYSERR    DD SYSOUT=*,OUTPUT=(*.OUTDSC)
//STDOUT    DD SYSOUT=*,OUTPUT=(*.OUTDSC)
//STDERR    DD SYSOUT=*,OUTPUT=(*.OUTDSC)
//SYSOUT    DD SYSOUT=*,OUTPUT=(*.OUTDSC)
//CEEDUMP   DD SYSOUT=*,OUTPUT=(*.OUTDSC)
```

The important parameters here are:

-r /web/httpd.conf - the server configuration file

_CEE_ENVFILE - points to where the server environment variables are set

More on these parameters shortly.

Modify the above to meet your needs and save it as member HTTPD1 in a PROCLIB in the system proclib concatenation.

Copy Configuration Files

The program files for the server come to some users in read-only files. That's OK, since we won't be modifying any executables.

But we do have to arrange for our configuration files to be read / write so we can modify the configuration. We need to have our logs and reports be in read / write directories so we can add to these as needed.

The Install doc talks about running a setup script, setup.sh. But there are problems with this. Mainly you have no control over where this script puts things. We find it works just fine to simply copy over a few files. This requires you to get into omvs again. So carefully follow these steps:

From ISPF 6 get into omvs:

```
    ===> omvs
```

once you are in omvs, issue this string of commands, one at a time:

```
su                        <- note: you do have to be superuser
cp /usr/lpp/internet/samples/config/C/h* /web
cp /usr/lpp/internet/samples/config/C/i* /web
cp /usr/lpp/internet/samples/config/C/j* /web
cp /usr/lpp/internet/samples/config/C/l* /web
cd /web
mkdir logs
mkdir reports
mkdir pub
cp /usr/lpp/internet/server_root/pub/C/*.html /web/pub
chmod 766 *
exit
exit
```

Hint for the 'cp' commands: function key F12 recalls the previous command, so you only need to arrow over to the letter to be changed and press <Enter>.

Modify Configuration Files

There are just tons of parameters to possibly modify, but here again we are just after the quick and dirty, minimal changes. So once again get into omvs from ISPF 6

```
        ===>  omvs
```

Then follow these steps

* get to the web directory: **`cd /web`**
* edit the httpd.conf file: **`oedit httpd.conf`**

    Note: <u>if you get permission denied, issue the su command then re-issue the oedit command</u>

    Note: oedit is essentially ISPF edit, so it should be plenty familiar. This is a large file so you will want to do 'find' commands; the changes here are in the order we expect to find lines in the file

f Port 1	if you changed the port number in the TCPIP parms, match it here
f PidFile 1	this value should be **/web/httpd-pid**
f AccessLog 1	this value should be **/web/logs/httpd-log**
f ErrorLog 1	this value should be **/web/logs/httpd-errors**
f CgiErrorLog 1	this value should be **/web/logs/cgi-error**
f AccessReportRoot 1	value should be **/web/reports**
f sslmode 1	ensure value is **off**
==> **f pass last 1**	for **Pass /*** the value should be **/web/pub/***

    F3 - end; exit the edit session

* leave omvs (one or two **exit** commands)

Start Up the HTTP Server

Now, go to SDSF and fire up the server:

```
===> /s httpd1
```

Finally, bring up your browser and point to your system:

http://www.*domain_name* (the domain for your system)

or

http://www.IP_address (your system IP address)

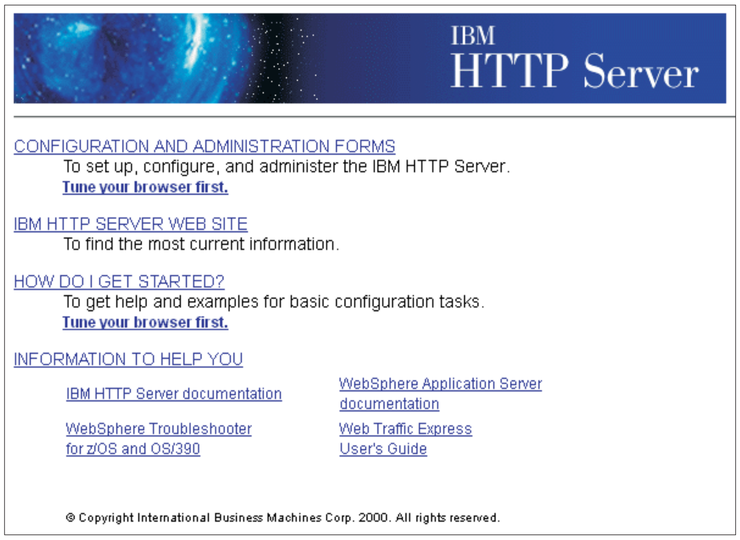
Note: you may need to append a port number, if you are using a port other than 80, for example:

http://www.*domain_name***:8081**

or

http://www.*IP_address***:8081**

You should be prompted for a user id and password and then your browser should show:



CONFIGURATION AND ADMINISTRATION FORMS
To set up, configure, and administer the IBM HTTP Server.
Tune your browser first.

IBM HTTP SERVER WEB SITE
To find the most current information.

HOW DO I GET STARTED?
To get help and examples for basic configuration tasks.
Tune your browser first.

INFORMATION TO HELP YOU

IBM HTTP Server documentation

WebSphere Application Server documentation

WebSphere Troubleshooter for z/OS and OS/390

Web Traffic Express User's Guide

© Copyright International Business Machines Corp. 2000. All rights reserved.

Making Adjustments

One reader told me the files I mentioned were not found in the same places on his system as on ours. To get around this, he had to make some other changes in httpd.conf. In particular, where there were **Pass** directives for Admin or admin-bin he had to change these by adding /C at the end. That is:

Change:	/usr/lpp/internet/server_root/Admin/*.jpg
To:	/usr/lpp/internet/server_root/Admin/C/*.jpg
Change:	/usr/lpp/internet/server_root/Admin/*.gif
To:	/usr/lpp/internet/server_root/Admin/C/*.gif
Change:	/usr/lpp/internet/server_root/Admin/*.html
To:	/usr/lpp/internet/server_root/Admin/C/*.html
Change:	/usr/lpp/internet/server_root/admin-bin/webexec/*
To:	/usr/lpp/internet/server_root/admin-bin/webexec/C/*

also:

Change:	/usr/lpp/internet/server_root/Docs/*
To:	/usr/lpp/internet/server_root/Docs/C/*

Making Adjustments, 2

If you find you need to change some of the httpd.conf values, the simplest way to have the new values take effect is to stop the server, wait for the stopping to end, then restart the server.

From an SDSF panel:

```
        ===> /p httpd1
```

(wait for it)

```
        ===> /s httpd1
```

One reader had problems due to file ownership of the server files. We were able to fix the problem by getting into omvs and issuing:

```
   ==> cd /usr/lpp/internet/server_root/admin-bin
   ==> chown WEBADM:IMWEB *out
```

If your installation has activated the RACF APPL class, and you have defined a profile for OMVSAPPL in the APPL class, then IDs that need to access OMVS should be permitted READ access to this profile.

Conclusion

There is plenty more work to do, as occasions and needs arise, but now you should have a functioning HTTP server running on your z/OS system.

The biggest stumbling blocks to getting all this done seem to be security issues, so it would be a good idea to befriend your local security administrator. The second level of problems come from network issues, so befriend your local network team.

Once you're off and running, however, there's no stopping you!

Enjoy!

Available Training

Of course, now you have the bulk of the setup necessary for running our z/OS UNIX web-related courses:

Introduction to z/OS UNIX - 3 days; details at:

http://www.trainersfriend.com/UNIX_and_Web_courses/u510descr.htm

Shell Script Programming in z/OS UNIX - 3 days; details at:

http://www.trainersfriend.com/UNIX_and_Web_courses/u515descr.htm

You and z/OS and the World Wide Web - 5 days; details at:

http://www.trainersfriend.com/UNIX_and_Web_courses/u518descr.htm

Introduction to CGIs on z/OS - 1 day; no labs; details at:

http://www.trainersfriend.com/UNIX_and_Web_courses/uc01descr.htm

Writing z/OS CGIs in COBOL - 2 days; details at:

http://www.trainersfriend.com/UNIX_and_Web_courses/uc04descr.htm

Writing z/OS CGIs in Assembler - 2 days; details at:

http://www.trainersfriend.com/UNIX_and_Web_courses/uc06descr.htm

Finally, other courses along these lines can be developed at your request.