



**The Trainer's Friend**  
I N C O R P O R A T E D

## **Porting Apache 2.2.9 to z/OS 1.9**

- ◆ 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](mailto:steve@trainersfriend.com)

v2.3

The following terms that may appear in this paper are trademarks or registered trademarks:

Trademarks of the International Business Machines Corporation: WebSphere, z/OS, z/Architecture, zSeries

Trademarks of Microsoft Corp.: Microsoft, Windows, Windows NT, Visual Basic, Microsoft Access, MS-DOS, Windows XP

Registered Trademark of Lenovo: ThinkPad

Registered Trademark of The Open Group: UNIX

Trademark of Sun Microsystems, Inc.: Java

Registered Trademark of Linus Torvalds: LINUX

Registered Trademark of Unicode, Inc.: Unicode

Trademarks held on behalf of World Wide Web Consortium: W3C, XHTML, XSL, WebFonts

Trademarks of the Apache Software Foundation: ASF, Apache.

## **Acknowledgements**

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

**Greg Ames, IBM**

**Kelly Arrey, IBM**

**John Boylston, IBM**

**Hunter Cobb, The Trainer's Friend, Inc.**

**Anthony Giorgio, IBM**

**Milos Lalovic, IBM**

**Bill Seubert, IBM**

**Mark Zelden, Zurich Insurance of North America**

**Special appreciation to Greg Ames, whose knowledge and insight were especially helpful.**

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

## Porting Apache 2.2.9 to z/OS 1.9

- The Goal
- Our Approach
- Locating Resources
- Setting Up the ID to do the Install
- Setting up zFS files
- Obtaining Necessary Files
- Installing Support Files
- Setting Environment Variables
- Extracting the Apache Files
- Installing libtool
- Running the patch File
- Configuring Apache
- Preparing and Running the Makefiles
- Modifying httpd.conf
- Testing Apache
- Troubleshooting
- Running Apache from JCL
- When to Use the Original HTTP Server or Apache

# The Goal

The Internet, specifically the subset of the Internet called the World Wide Web ("www", or just the Web), is the platform of choice for most businesses today. So it seems natural to want to support the Web from the mainframe (specifically, from the z/OS software platform, running on z Architecture machines).

IBM provides three basic approaches to this:

- \* A free web server is included with z/OS
- \* The latest versions of CICS (CICS/TS) can serve pages to the web
- \* You can purchase the WebSphere Application Server (WAS) product

But, of course, there are tradeoffs:

- \* The free web server, while servicable for many applications, is based on older technology (the original CERN server); it will not be enhanced and it does not include support for newer features such as IPv6 and 64-bit addressing
- \* Not everyone has CICS/TS, and new development may well be better served by using direct connection to the Web instead of through CICS
- \* The WAS product is quite expensive, with a core product and a number of optional, priced features; it also has a large "footprint", requiring lots of disk space, memory, and cycles

Furthermore, the most commonly used web servers today are based on the Apache HTTP server from the open source Apache Software Foundation, and anyone who wants to keep abreast of current Web practices will need an Apache-based web server.

So, we cast about for ways to get an Apache server on z/OS. Here there seem to be only two options:

- \* If you have WebSphere, the latest releases provide an Apache based server called "The IBM HTTP Server for z/OS Powered by Apache"

Downside: see comment above about WAS

- \* You can port the actual Apache server from the Apache Software Foundation to z/OS; it's free.

Downsides: no support from IBM, and it takes some effort to do

## The Goal, continued

The purpose of this paper is to lay out the process of porting the latest version of the Apache web server to z/OS. All kinds of caveats apply, including:

- \* This work demonstrates the process used for a very small environment (a mainframe on a PC; long story)
- \* Minimal work was done regarding security of the network and the server itself: the goal is to get the functionality working
- \* There has been no large volume stress test

Nonetheless, the interested reader may find this paper helps him or her over the natural hurdles we encountered in successfully porting the Apache server to a z/OS system.

The version of Apache we worked with is 2.2.9, and we ported this to a z/OS system at level 1.9 (that is, Apache 2.2.9 ported to z/OS 1.9). The instructions here may work for other versions, but there are no guarantees. Note that the version of the Apache server shipped with WebSphere is in WebSphere 6.1.

The Open Source world is a dynamic, fast changing one; at one point in the process we found one component was obsolete because it was two weeks out of date! (To be fair, we caught things in transition, at a release level.)

The process here should provide you with an Apache server that is stable for many years, as long as you don't need any features introduced in later releases. In any event, we hope you find these notes helpful in getting your own Apache server up and running quickly, with a minimum of difficulties.

While working on this project we went down many blind alleys and did things a more experienced UNIX administrator would have done differently. In this paper we have simply laid out a near-cookbook approach that summarizes an approach that will lead you to a successful installation of a basic Apache server running under z/OS.

# Our Approach

In this paper we organize our notes along these lines:

- \* **Locate information - find places where program files and supporting documentation can be located**
- \* **Prepare for the installation - make decisions covering these areas:**
  - + **Where to put a directory on your workstation, and what to call it**
  - + **What userid will you use to do the installation work on z/OS**
  - + **What directory structure will you use: where to place the Apache server and supporting tools**
- \* **Download needed files to your workstation**
- \* **Set up z/OS RACF info for userid (or ensure planned ID has the required attributes); of course, other security packages may be used**
- \* **Allocate disk space, build and mount directories needed**
- \* **Upload files to correct locations on your z/OS system**
- \* **Install support files, create support scripts**
- \* **Run the build and configure steps for Apache**
- \* **Configure Apache run-time directives**
- \* **Test Apache, get it to run the way you need**
- \* **Setup JCL to run Apache as a started task**

**We will use the names and values we used in our work (changed where there might be security issues, of course).**

**When we give instructions for z/OS UNIX work, we will show all the commands for each step, even though some may be redundant: we found we could not do it all in one sitting, so we tried to ensure the command strings will always place you at the right directory in the right mode.**

# Locating Resources

None of us can do it all these days. Life, in particular Information Technology, is too rich and complex to "get" it all. So we went hunting for resources to help us in our quest, and here are the primary ones we used:

1. The most promising pages seemed, at first, to be the pages titled "Apache 2.0 on OS/390" and "Apache 2.0 for z/OS" at

<http://people.apache.org/~trawick/apache-2-on-os390.html>

and

<http://people.apache.org/~trawick/apache-2-on-zos.html>

(Note: these two pages are the same page, with only a different <title> tag!) They are considerably out of date; however, one of the page authors, Greg Ames, provided invaluable assistance in our efforts here.

2. You need to have the **C/C++ compiler** for z/OS available; this is a fee product your installation must have to do this work

3. The IBM page for z/OS UNIX Tools and Toys, containing ported tools. At first, we thought about only getting the specific tools we would need, as we found out we needed them; this turned out to be a mistake. You can download and install the complete collection at one time and then you don't have to worry. Start at:

<http://www-03.ibm.com/servers/eserver/zseries/zos/unix/bpxa1ty1.html>

We'll take one particular file from here, then...

4. You can get the whole Ported Tools package at:

<http://www-03.ibm.com/servers/eserver/zseries/zos/unix/redbook/index.html>

We will get the whole package shortly.

## Locating Resources, 2

5. You will need special versions of several programs; for these go to:

<http://people.apache.org/~gregames/2.2-zOS-build/>

This page on the Apache site is maintained by Greg Ames of IBM.

6. The site for the Apache Software Foundation (ASF) is:

<http://apache.org/>

This page has links to a variety of projects under way at ASF, and the first one is for the HTTP Server - that's the one we want. If you want to just start there, the link is:

<http://httpd.apache.org/>

You will want to look around this site for supporting documentation and tutorials.

7. If you are interested in the WAS version of Apache, you can get a doc at

<http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP101170>

and even though it's for the WAS environment, it makes good reading and points out some z/OS things that are not obvious from the other resources

8. Other resources include the listserv's for mvs-oe and ibm-main, both for the direct give and take and for the archive features. These groups were excellent help for the z/OS side of the work.

9. Finally, we found the Apache HTTP Server mailing list to be invaluable in helping with the Apache-specific problems and issues. (Actually, we subscribed to the mailing list for a related product, APR (Apache Portable Runtime) that is used by the HTTP server and other programs; the people there were terrific.) This group helped us with the Apache part of the project.

# Setting Up the ID to do the Install

You may have an ID that is all ready to go, but we ran into a problem and had to fix ours, so it's best to ensure you have the needed attributes before you begin:

- \* TSO ID needs to have large default and maximum region sizes; we ended up specifying 2048000 and 2096128, respectively
- \* UNIX ID needs to have superuser authority and a large region; we set our defaults (in BPXPRMxx) this way:

```
MAXASSIZE(2147483647)
MAXPROCSYS(2000)
MAXPROCUSER(250)
MAXUIDS(200)
MAXFILEPROC(10000)
MAXPTYS(256)
MAXSHAREPAGES(524288)
MAXTHREADTASKS(5000)
MAXTHREADS(10000)
MAXCPUTIME(7200)
RESOLVER_PROC(RESOLVER)
SUPERUSER(OMVSKERN)
CTRACE(CTIBPX00)
```

The above are for all z/OS UNIX users, of course; our particular ID had these z/OS UNIX settings in its OMVS segment:

```
UID= 0000000021
HOME= /u/admins
PROGRAM= /bin/sh
CPUTIMEMAX= NONE
ASSIZEMAX= NONE
FILEPROCMAX= NONE
PROCUSERMAX= NONE
THREADSMAX= NONE
MMAPAREAMAX= NONE
```

## Setting up zFS Files

We wanted to set up a separate zFS file for our Apache files, and one for our tools and toys files, so we created separate zFS files and established mount points so that when we get our next release of z/OS we can simply move these to the new system.

We get a pre-generated z/OS system which we then tailor to preserve our settings and files across new version installations. We prepare for this by building files on volumes that don't get overlaid as we install each new z/OS.

The jobs we ran to accomplish this (could be a single job, of course):

```
//ZFSAPACH JOB , 'ZFS ALLOCATE', NOTIFY=&SYSUID,
//          USER=IBMUSER, PASSWORD=xxxxxxx,
//          CLASS=A, MSGCLASS=X, MSGLEVEL=(1,1)
// SET JOBID=ZFSAPACH
//*
//* This job defines and formats a ZFS data set to hold the
//* Apache web server
//*
//DEFINE   EXEC   PGM=IDCAMS
//SYSPRINT DD   SYSOUT=*
//SYSIN    DD
//          DEFINE CLUSTER (NAME(MANAGE.APACHE.ZFS) -
//                          VOLUMES(ZFSTUF) -
//                          LINEAR CYL(600 40) SHAREOPTIONS(3))
/*
//CREATE   EXEC   PGM=IOEAGFMT, REGION=OM,
// PARM=('-aggregate MANAGE.APACHE.ZFS -compat')
//SYSPRINT DD   SYSOUT=*
//STDOUT   DD   SYSOUT=*
//STDERR   DD   SYSOUT=*
//CEEDUMP  DD   SYSOUT=*
```

```
//ZFSTOYS JOB , 'ZFS ALLOCATE', NOTIFY=&SYSUID,
//          USER=IBMUSER, PASSWORD=xxxxxxx,
//          CLASS=A, MSGCLASS=X, MSGLEVEL=(1,1)
// SET JOBID=ZFSTOYS
//*
//* This job defines and formats a ZFS data set to hold
//* the ported tools and toys programs from IBM
//*
//DEFINE   EXEC   PGM=IDCAMS
//SYSPRINT DD   SYSOUT=*
//SYSIN    DD
//          DEFINE CLUSTER (NAME(MANAGE.TOOLS.ZFS) -
//                          VOLUMES(ZFSTUF) -
//                          LINEAR CYL(400 20) SHAREOPTIONS(3))
/*
//CREATE   EXEC   PGM=IOEAGFMT, REGION=OM,
// PARM=('-aggregate MANAGE.TOOLS.ZFS -compat')
//SYSPRINT DD   SYSOUT=*
//STDOUT   DD   SYSOUT=*
//STDERR   DD   SYSOUT=*
//CEEDUMP  DD   SYSOUT=*
```

## Setting up zFS Files, 2

We then had to get into OMVS, make these directories, and mount the files at the correct point:

```
su
cd /usr/lpp
mkdir zApache
mkdir toolsNtoys
/usr/sbin/mount -t zfs -f MANAGE.APACHE.ZFS /usr/lpp/zApache
/usr/sbin/mount -t zfs -f MANAGE.TOOLS.ZFS /usr/lpp/toolsNtoys
```

In addition, you will want to update your BPXPRM<sub>xx</sub> member(s) in your PARMLIB concatenation for later IPLs to bring these files online automatically. We added these lines:

```
MOUNT      FILESYSTEM( 'MANAGE.APACHE.ZFS' )
           TYPE(ZFS)
           MODE(RDWR)
           MOUNTPOINT( '/usr/lpp/zApache' )

MOUNT      FILESYSTEM( 'MANAGE.TOOLS.ZFS' )
           TYPE(ZFS)
           MODE(RDWR)
           MOUNTPOINT( '/usr/lpp/toolsNtoys' )
```

**Note:** if you are able to unwind the tools and toys package right into /usr/local, you can omit the second mount in both places.

## Setting up zFS Files, 3

We mentioned this on `ibm-main`, and got the following good advice from Mark Zelden:

Most sane shops would need to do this to a maintenance root mounted at a service mount point, clone, and re-IPL since the root is usually mounted read only and may be shared in a non-sysplex aware file system environment. Even in a shared file system environment the `sysres` root should be mounted read only.

Example:

```
su
cd /service/usr/lpp
mkdir zApache
```

-----

Now, it really doesn't matter in our small shop, but if you work in a larger shop you should place close attention to Mark's point.

A point about naming: The names we chose for the directories seemed like a good idea at the time, but we later learned that a more common (but not universal) convention would be to put the two directories under the `/usr/local` mount point (that is, `/usr/local/zApache` and `/usr/local/toolsNtoys`). But, we didn't, and we got things to work anyway, but maybe it was a little more difficult this way.

Note also, we uploaded all files to a special directory we built under our ID, `/u/admins/apache` - so we could simply copy them under `z/OS UNIX` if we needed to re-run or re-try any step.

## Obtaining Necessary Files

(We initially put all the downloaded files into E:\Apache-for-zOS on our workstation.)

Now, download the ported tools binaries from the IBM site for ported tools and toys mentioned on page 8. They come down in binary, just save them on your workstation.

On <http://www-03.ibm.com/servers/eserver/zseries/zos/unix/bpxa1ty1.html>  
First scroll down and locate autoconf (v2.54); click on that and download this file to your workstation (the link is for "[autoconf-2.54-ebcdic-bin.pax](#)", and we'll need this).

Then, click on any of the lines that read:

This package is part  
of the [Redbook](#)  
["Open Source](#)  
[Software" distribution.](#)

this takes you to:

<http://www-03.ibm.com/servers/eserver/zseries/zos/unix/redbook/index.html>

Under "Package", the row labeled "All packages, 2001 redbook", click on the "[binary](#)" link, which will ask you where to save "[5944-01.binaries.tar.Z.bin](#)". This is the entire collection of ported tools.

[Even though the package includes an autoconf, we need to have the separate version mentioned above so we can install it in the expected location.]

Note: do not get too excited when you see "Apache" in the list; one of the links is broken, the other just takes you back to the Apache on z/OS page we referenced earlier.

Also note: there is a link to a PDF version of the related RedBook (documentation) that might be good to get.

## Obtaining Necessary Files, 2

Next we need to get special versions of some standard UNIX tools, since the GNU version doesn't work well with z/OS UNIX shared objects, and some of the standard distribution files need to be patched to generate code that is z/OS aware.

Go to <http://people.apache.org/~gregames/2.2-zOS-build/>. There are four files you should download:

**build.patch** - a text file

**libtool-zOS-httpd-2.2.tar.Z** - a binary file

**my\_config** - a text file

**my\_envs** - a text file

Click on the binary file and you will be prompted to open or save; select Save to disk, and when prompted, specify your workstation directory.

Click on each of the text files and the text will display in front of you in a browser page. Right click on any blank spot on the page and a pop up menu gives you a list of choices, one of which is Save page as ... - if you select this you get a browse type dialog where you can point to your workstation directory.

## Obtaining Necessary Files, 4

Next download the Apache server file from the Apache site.

Start with the Apache site: [apache.org](http://apache.org), which has a link to the httpd project (<http://httpd.apache.org>) which contains the following links:

Download

( <http://httpd.apache.org/download.cgi> )

Documentation, Version 2.2

( <http://httpd.apache.org/docs/2.2/> )

The Download link page contains several choices, including:

Apache HTTP Server 2.2.9 is the best available version and a variety of files to download. Select [httpd-2.2.9.tar.gz](#) (the Unix version that is gzip'ped)

You might want to look at the documentation link and download or bookmark some of these pages, too.

Then upload from your workstation to your z/OS id directory (/u/admins/apache in our case) then copy them into these directories:

* <a href="#">httpd-2.2.9.tar.gz</a>	into /usr/lpp/Apache
* <a href="#">libtool-zOS-httpd-2.2.tar.Z</a>	into /usr/lpp/zApache
* <a href="#">build.patch</a>	into /usr/lpp/zApache
* <a href="#">my_config</a>	into /usr/lpp/zApache
* <a href="#">my_envs</a>	into /usr/lpp/zApache
* <a href="#">5944-01.binaries.tar.Z.bin</a>	into /usr/lpp/toolsNtoys
* <a href="#">autoconf-2.54-ebcdic-bin.pax</a>	into /usr/lpp/toolsNtoys

Remember: upload all the tar files and the pax file in binary mode, the others in text mode. [If you can, copying the last two files into /usr/local might make your life easier; we did not really have that option.]

# Installing Support Files

Before we attack the Apache file, we need to install the tools.

The tools and toys install is simple:

```
su                                <== if not already su
cd /usr/lpp/toolsNtoys           <== start in right place
tar -x -f 5944-01.binaries.tar.Z.bin
```

This creates an entire directory hierarchy and now the tools are available. It takes a while to run, so be patient.

We also need to install the special autoconf we obtained:

```
su                                <== if not already su
cd /usr/lpp/toolsNtoys           <== if not already there
pax -rf autoconf-2.54-ebcdic-bin.pax <== unwinds archive
```

This autconf is hard coded to be in /usr/local, so this is where the pax places the files it unwinds

[Again, the above two steps may be run in /usr/local if you can, and things will be a little simpler.]

Now is a good time to create a small script we'll need later. Use oedit or vi to create a script named which consisting of these two lines:

```
#!/bin/sh
whence $*
```

This script should be placed into the /usr/local/bin directory

## Installing Support Files, 2

Now, we need to set up some links and environment variables to ensure our programs are found correctly:

<code>su</code>	<code>&lt;==</code> if not already su
<code>cd /usr/local</code>	<code>&lt;==</code> where files are expected
<code>ln -s /usr/lpp/toolsNtoys/lib lib</code>	<code>&lt;==</code> link for lib files
<code>cd bin</code>	<code>&lt;==</code> get to /usr/local/bin
<code>ln -s /usr/lpp/toolsNtoys/bin/perl perl</code>	<code>&lt;==</code> access perl in toolsNtoys
<code>ln -s /usr/lpp/toolsNtoys/bin/m4 m4</code>	<code>&lt;==</code> access m4 in toolsNtoys

[If you have unwound everything under /usr/local you can skip this step.]

## Setting Environment Variables

To save time, we created a script named tempsetup in /etc that looked like this:

```
PATH=/usr/local/bin:$PATH
PATH=$PATH:/usr/lpp/zApache/bin
PATH=$PATH:/usr/lpp/toolsNtoys/bin
export PATH
export LIBTOOL_PREFIX=/usr/lpp/zApache

export _CC_CCMODE=1
export CC=cc
export CFLAGS="-Wc,XPLINK,dll,expo -WI,XPLINK -O3"
export LDFLAGS=" -L/usr/lpp/zApache/httpd-2.2.9/src/lib/apr
                -L/usr/lpp/zApache/httpd-2.2.9/src/lib/apr-util "
export APRUTIL_LDFLAGS=-L/usr/lpp/zApache/httpd-2.2.9/built/lib
```

This script was based on the **my\_envs** file and we added some configuration variations for our environment.

**Important note:** the location of our autoconf (/usr/local/bin) and our special libtool (/usr/lpp/zApache/bin) have to appear before the location of the toolsNtoys files in our PATH environment variable. [If you were able to unwind the tools and toys and the autoconf under /usr/local, you can omit the third line above.]

Even though you don't need all these values at this point in time, this should save you some time and effort in the long run. The LDFLAGS value will fit on one line

After you've installed Apache, you can extract the parts of this you'll be using every day into your /etc/profile script, so they are available to everyone, or in your private .profile script for testing and evaluation. We anticipate you'll want the PATH lines but that's probably all, once Apache is running.

For now, every time you get into OMVS by ISPF or telnet, issue this first:

```
. /etc/tempsetup          <== note a space between "." and "/"
```

Only run this once per session.

## Extracting the Apache Files

We begin with the program we downloaded called `httpd-2.2.9.tar.gz` (see page 15) in our `zApache` directory:

<code>su</code>	<code>&lt;== if not su already</code>
<code>cd /usr/lpp/zApache</code>	<code>&lt;== ensure in our Apache directory</code>
<code>gzip -d httpd-2.2.9.tar.gz</code> (this creates <code>httpd-2.2.9.tar</code> and removes <code>httpd-2.2.9.tar.gz</code> )	<code>&lt;== uncompress; uses gzip from the ported tools we installed</code>
<code>pax -rf httpd-2.2.9.tar -o to=ibm-1047</code>	<code>&lt;== this creates subdirectory httpd-2.2.9 and converts all text files to EBCDIC from ASCII</code>

Next, we have some more prep to do ...

# Installing libtool

Apache, GNU, and the rest of the open source community have evolved a complex set of programs and files to automate building applications on any platform supporting UNIX type constructs. The details are not necessary here, but this does mean we have to build some uniquely z/OS versions of some of these programs and files. Greg Ames has provided them on the site we discussed earlier.

First: libtool; follow this string of commands:

```
su                                <== if not already

cd /usr/lpp/zApache               <== get to starting directory

pax -rf libtool-zOS-httpd-2.2.tar.Z -o to=ibm-1047
                                   <== unzip and unwind the tar file
```

This creates a directory hierarchy beginning with a directory libtool; that is: /usr/lpp/zApache/libtool

```
cd libtool                        <== get into the libtool directory
```

(at this time, you need to have the LIBTOOL\_PREFIX environment variable set to /usr/lpp/zApache (or, of course, your version of it) set; see page 18)

```
make                              <== prepare for libtool install
```

```
make install                      <== install libtool
```

Sanity check: issue this command

```
libtool --version
```

and you should see:

```
libtoolexe: This is libtool 1.3.9 for z/OS.
It acts enough like GNU libtool to allow Apache to be built.
```

## Running the patch File

Next, we need to run the `build.patch` which modifies some of the Apache setup scripts. First, copy `build.patch` into the Apache directory, then run the patch command in the `httpd-2.2.9` sub-directory:

<code>su</code>	<== if not already su
<code>cd /usr/lpp/zApache</code>	<== get to zApache directory
<code>cp build.patch httpd-2.2.9</code>	<== copy patch file
<code>cd httpd-2.2.9</code>	<== get to httpd-2.2.9 directory
<code>patch -p0 -i <b>build.patch</b></code>	<== run the patch command

This should produce a series of messages such as:

```
Hmm... Looks like a new-style context diff to me...
The text leading up to this was:
-----
!*** ./srclib/apr/buildconf.orig      Tue Jul  8 13:56:00 2008
!--- ./srclib/apr/buildconf          Wed Jun 25 15:45:37 2008
.
.
.
-----
Patching file ./srclib/pcre/configure.in using Plan A...
Hunk #1 succeeded at 7.
done
```

# Configuring Apache

The next step is to prepare to configure the Apache files. This entails running a script called `buildconf` :

```
su                <== in not already su
cd /usr/lpp/zApache/httpd-2.2.9  <== get to Apache directory
./buildconf       <== run the script
```

This task ran a long time on our small system (about five and a half hours). we recommend that you run it under telnet instead of OMVS. Remember to run `/etc/tempsetup` at the start of your telnet session.

You are likely to see messages like:

```
WARNING: Using auxiliary files such as `acconfig.h', `config.h.bot'
WARNING: and `config.h.top', to define templates for `config.h.in'
WARNING: is deprecated and discouraged.
```

```
WARNING: Using the third argument of `AC_DEFINE' and
WARNING: `AC_DEFINE_UNQUOTED' allows to define a template without
WARNING: `acconfig.h':
```

```
WARNING: AC_DEFINE([NEED_MAIN], 1,
WARNING:           [Define if a function `main' is needed.]
```

```
WARNING: More sophisticated templates can also be produced, see the
WARNING: documentation.
```

These are just "nuisance messages". you can ignore them.

**NOTE:** for our work, `/usr/lpp/zApache/httpd-2.2.9` is known as the Apache prefix. This comes up a number of times. Be sure to use your naming conventions, of course, whenever the Apache "prefix" is referenced; notice that it is all directories in the path up to the directory in which Apache was unwound.

## Configuring Apache, 2

**Next, if buildconf is successful we run configure. This script tests various features of the system. In some cases it dynamically builds programs and compiles them; if an error is detected, it is intercepted; that tells configure something about your system. After gathering all information it can, it builds the source files for your server.**

**The configure script invokes similar configure scripts in subdirectories, and these scripts run similar tests. This can take a long time to run: on our system, the configure script ran for over six hours! (Remember, we were running z/OS on top of flex-es on top of Linux on a ThinkPad; so we had a lot of levels of emulation going on.) This is another good choice for a task to run via telnet.**

**We wanted to get all the related information for this step on one page, so see the next page for the details ...**

## Configuring Apache, 3

Now, `configure` has many options, and we recommend you explore them and experiment with them as time and interest permit. In fact, we recommend you put your `configure` command in a script with all the options you need. That is what the `my_config` file is for.

As it comes from Greg Ames page, `my_config` looks like this:

```
./configure --prefix=$PWD/built --with-expat=builtin \  
--enable-mods-shared=most --disable-dav --with-mpm=worker --with-included-apr \  
--libdir=$PWD/built/lib -C
```

After tailoring to work for our environment it looks like this:

```
./configure --prefix=/usr/lpp/zApache/httpd-2.2.9/built --with-expat=builtin \  
--enable-mods-shared=most --disable-dav --with-mpm=worker --with-included-apr \  
--with-port=8080 --libdir=/usr/lpp/zApache/httpd-2.2.9/built/lib -C
```

Note we set the prefix explicitly and we chose port 8080 because we wanted to continue to run the old HTTP server listening on port 80. At the time we copied `my_config` to our files, it went to `/usr/lpp/zApache` because we didn't have the later subdirectories available. Now, it's probably best to modify `my_config` then copy it into `httpd-2.2.9`. So here's the run of `configure`:

<code>su</code>	<== if not already su
<code>cd /usr/zApache</code>	<== pick up <code>my_config</code>
<code>cp my_config httpd-2.2.9</code>	<== put into right place
<code>cd httpd-2.2.9</code>	<== get into this subdirectory
<code>. my_config</code>	<== run the script

The prefix value specifies the directory to contain the server and its directory structure (note the `/built` sub-directory); this value is prefixed to all the directory and file names generated by the `configure` process; the `-C` says to cache test results; this should speed things up for the nested `configure` scripts, avoiding repeating a number of tests.

## Preparing and Running the Makefiles

If all has gone well, the next step is to run `make` and then, when that's done, `make install`:

<code>su</code>	<code>&lt;== if needed</code>
<code>cd /usr/lpp/zApache/httpd-2.2.9</code>	<code>&lt;== this is where the files are</code>
<code>make</code>	<code>&lt;== this builds the base files</code>

The above step is another long running one (over 3 hours), so consider doing it under `telnet`. Then we do the actual product install:

<code>make install</code>	<code>&lt;== this runs the installation process</code>
---------------------------	--

At this point, the Apache server is installed in `/usr/lpp/zApache/httpd-2.2.9/built`. This subdirectory has two lower subdirectories we care about: `conf` which holds the configuration files (mostly `httpd.conf`, discussed shortly) and `bin` which holds the executables and scripts we need to start Apache running.

There's still some tidying up to do. First, a couple of files that the `make / make install` process should copy into one of our libraries don't get copied over, so we'll take care of that:

<code>su</code>	<code>&lt;== in case</code>
<code>cd /usr/lpp/zApache/httpd-2.2.9</code>	<code>&lt;== starting point</code>
<code>cp srclib/apr/libapr-1.so built/lib</code>	<code>&lt;== copy</code>
<code>cp srclib/apr-util/libaprutil-1.so built/lib</code>	<code>&lt;== copy</code>

-----

It seems there can be some spurious results if you don't set permissions of a few files, so just in case you should issue these commands once:

<code>su</code>	<code>&lt;== in case</code>
<code>cd /usr/lpp/zApache/httpd-2.2.9/built/logs</code>	<code>&lt;== starting point</code>
<code>touch access_log error_log httpd.pid</code>	<code>&lt;== create files</code>
<code>chmod 666 access_log error_log httpd.pid</code>	<code>&lt;== set permissions</code>

# Modifying httpd.conf

Now, there's just a little bit more to do before we test. The httpd.conf file will have to be modified to reflect our needs (some of this will be already taken care of, but there are still some things that don't get picked up in the configure and build process. So...

```
su                                     <== just in case
cd /usr/lpp/zApache/httpd-2.2.9/built/conf   <== get to where the file is

-- now use vi or oedit to modify httpd.conf as follows --
```

The contents of httpd.conf are called directives. Some things to do / consider...

1. Make sure the Listen directive specifies the port you are planning to use:

```
Listen 8080
```

2. Comment out the line:

```
LoadModule auth_digest_module modules/mod_auth_digest.so
```

**\*\*\* More \*\*\***

## Modifying httpd.conf, 2

3. After all the LoadModule directives, insert these lines to get translation between EBCDIC and ASCII:

```
<Directory />  
  
    CharsetSourceEnc IBM-1047  
    CharsetDefault  ISO8859-1  
  
</Directory>
```

Actually, we do a fair amount of work where we emit XHTML encoded in utf-8 and utf-16; under the HTTP server that comes with z/OS we had set up a naming convention that text files we did not want to be translated had a file name suffix of .ascii. we found we could reproduce this behavior under Apache by using this set of directives:

```
<Directory />  
  
    <FilesMatch "\.ascii$">  
        CharsetSourceEnc ISO8859-1  
        CharsetDefault  ISO8859-1  
    </FilesMatch>  
  
    CharsetSourceEnc IBM-1047  
    CharsetDefault  ISO8859-1  
  
</Directory>
```

3. Next add the following directive (it's supposedly the default, but we seemed to need it explicitly):

```
UserDir public_html
```

## Modifying httpd.conf, 3

4. Again, reflecting some naming conventions we used under the older HTTP server, for each developer using the server we provided two directives, one to recognize style sheets and where to find them, and one to identify CGI programs and where to find them; for example:

```
Alias    /s-css/    /u/scomsto/CGI/  
ScriptAlias /SCOMSTO/ /u/scomsto/CGI/
```

5. Comment out the User daemon and Group daemon directives (a comment is a leading #)

6. Assign values to the ServerAdmin and ServerName directives; turns out they don't have to be meaningful, just syntactically correct; we used:

```
ServerAdmin steve@trainersfriend.com  
ServerName StevesZServer
```

7. Instead of being closed to all, change this set of directives:

```
<Directory />  
Options FollowSymLinks  
AllowOverride None  
Order deny,allow  
Deny from all  
</Directory>
```

... to this, which will be open to all:

```
<Directory />  
Options FollowSymLinks  
AllowOverride None  
Order allow,deny  
Allow from all  
</Directory>
```

**IMPORTANT NOTE:** this latter is not secure, but probably OK for testing. Later you want to add directives to be more secure.

## Modifying httpd.conf, 4

8. This set of directives:

```
<IfModule dir_module>  
  DirectoryIndex index.html  
</IfModule>
```

is how you specify what files to look for if a path is specified without a file name; we changed this to more closely match what we used in the other server; so:

```
<IfModule dir_module>  
  DirectoryIndex index.html Welcome.html welcome.html  
</IfModule>
```

There are lots more possibilities, but the above let us test and work with the browser. Feel free to explore and experiment - but after you have the basic server working.

# Testing Apache

To test your installation requires two steps: 1) start the server and 2) point your browser to it.

From now on, you do not want to run your tempsetup script and you definitely do not want to be running as superuser. So consider getting out of any current telnet or omvs session you are in and re-logging in. Then ...

```
cd /usr/lpp/zApache/httpd-2.2.9/built/bin      <== where the action is
apachectl start                             <== provided script
```

To verify the task has started, you can issue:

```
ps -ef
```

and you should see one or more processes described something like this:

```
ADMINS  50398623      1  - 11:20:10 ?      0:00
          /usr/lpp/zApache/httpd-2.2.9/built/bin/httpd -k start
```

(each process shown all on one line).

Finally, bring up your browser and point it to your server using a URL like:

```
http://server_name:8080/
```

using your IP address or domain name and the port you chose; you should see a page in your browser that says:

**It works!**

If so, you're ready to start more full-scale testing of pages and programs that meet your needs and requirements.

# Troubleshooting

Just a few things that helped when we ran into problems:

\* Check out your `error_log` located in `prefix/built/logs` (in our case, `/usr/lpp/zApache/httpd-2.2.9/built/logs/error_log` )

\* You can stop Apache by:

```
cd /usr/lpp/zApache/httpd-2.2.9/built/bin      <== where the action is
apachectl stop                               <== provided script
```

After you stop Apache, if you make some changes and want to try it again, you can either do "apachectl start" again, or you can issue:

```
apachectl restart
```

\* If Apache doesn't seem to start, you may have to kill any lingering processes (use the PID(s) returned from the `ps -ef` command):

```
su                                             <== for this you must be su
kill -s KILL pid_number                      <== use obtained PID
```

You will also have to get rid of any socket files still around; they would be found in your logs directory; again, you have to run su for this:

```
su                                             <== if not already
cd /usr/lpp/zApache/httpd-2.2.9/built/logs   <== where you need to be
rm -f cgisock*                               <== remove socket files
```

Remember: you do not want to be running as su when you start, stop, or restart Apache, but you do want to run su for cleanup work.

# Running Apache from JCL

It would be nice to be able to start and stop Apache from outside omvs, so here we describe the pieces we needed to work together to accomplish this in our environment

We created a new RACF profile in the STARTED class which we named APACHE.\*\* using a RACF command like:

```
RDEFINE STARTED APACHE.** OWNER(xxxxx) UACC(NONE) -
STDATA(USER(ADMINS) GROUP(SYS1) TRUSTED(NO) PRIVILEGED(NO) TRACE(NO))
```

Followed by: `setr generic(started) raclist(started)`

Note that the user id in STDATA(USER( must not have UNIX UID(0)

We created a proc named APACHE in one of our SYSPROC concatenations, something like this:

```
//APACHE PROC ACTION='start',
// DIR='/usr/lpp/zApache/httpd-2.2.9/built',
// CONF='conf/httpd.conf'
//WEBSRV1 EXEC PGM=BPXBATCH,REGION=OK,TIME=NOLIMIT,
// PARM='PGM &DIR/bin/apachectl -k &ACTION -f &CONF ',
// MEMLIMIT=1024M
//STDOUT DD PATH='&DIR/logs/proc.output',
// PATHOPTS=(OWRONLY,OCREAT),PATHMODE=(SIRWXU,SIRWXG,SIRWXO)
//STDERR DD PATH='&DIR/logs/proc.errors',
// PATHOPTS=(OWRONLY,OCREAT),PATHMODE=(SIRWXU,SIRWXG,SIRWXO)
//SYSIN DD DUMMY
//OUTDSC OUTPUT DEST=HOLD
//SYSPRINT DD SYSOUT=*,OUTPUT=(*.OUTDSC)
//SYSERR DD SYSOUT=*,OUTPUT=(*.OUTDSC)
//SYSOUT DD SYSOUT=*,OUTPUT=(*.OUTDSC)
//CEEDUMP DD SYSOUT=*,OUTPUT=(*.OUTDSC)
```

Note: the server we built was all 31-bit, so you can omit the MEMLIMIT above; it is possible to build a 64-bit version of the server.

We used SDSF to enter operator commands, so:

```
COMMAND INPUT ==> /s apache
```

starts your started task. The task starts Apache, then ends (not visible in SDSF ST), but Apache continues to run (visible in SDSF DA).



# When to Use the original HTTP Server or Apache

The paper referenced earlier, at

<http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP101170>

also has a comparison of these two web servers and what criteria you might use to choose using one over the other. We intend to maintain both for the foreseeable future.

You might also find this report interesting:

[http://news.netcraft.com/archives/2008/07/07/july\\_2008\\_web\\_server\\_survey.html](http://news.netcraft.com/archives/2008/07/07/july_2008_web_server_survey.html)

Some people also suggested that there are likely to be a lot of people with Apache skills in the job market, and those skills can be easily carried over to the z/OS platform, whereas skills in developing for and supporting the older server might not be as widespread.

**Now it's time to read, do, experiment, make it work for you.**

**Please email corrections and suggestions to me at [steve@trainersfriend.com](mailto:steve@trainersfriend.com)**