Tracing Tips for WebSphere V4 for z/OS

This paper provides an overview of the various traces for diagnosing problems with WebSphere Application Server Version 4 or 4.0.1 for z/OS based on our experiences. For the latest version, check the "Hints and Tips" category at http://www.ibm.com/support/techdocs/ .

For detailed diagnosis information, see "*WAS V4 for z/OS Messages and Diagnosis"* which is downloadable from http://www.ibm.com/software/webservers/appserv .

WAS Error Log Stream – This contains error messages from the WAS runtime servers.

- Set-up: See the "Installation & Customization" Guide. The logstream name was specified when you installed WAS and is in the 'current.env' for each server instance, such as: `LOGSTREAMNAME=WAS.ERROR.LOG`
- To browse it, use the Log Browse Utility: `TSO BBORBLOG 'logstream_name'`
  - You may have to explicitly specify dataset name if BBORBLOG is not in your SYSEXEC library: `TSO exec 'WAS401.SBBOEXEC(BBORBLOG)' 'logstream_name'`
- This creates a dataset called "userid.logstream_name" and places you in ISPF browse mode of it.
  NOTE: Each time you invoke BBORBLOG, a static file is created which deletes the former version, so you have to re-issue the BBORBLOG to refresh it.
- BBORBLOG can also be invoked from the OMVS Shell. (See WSC Sample Programs below.)

JRAS – This is useful for debugging Java runtime servers.

- Set-up: Place the 'trace.settings' file in the same directory as the 'current.env' file, and point to it from the 'jvm.properties' file with the `com.ibm.ws390.trace.settings` parameter.
  - Sample entry in the 'jvm.properties' file (should be all one one line): `com.ibm.ws390.trace.settings=/WebSphere390/WAS401/controlinfo/envfile/WSLPLEX/BBOASR2A/trace.settings`
  - Sample contents of "trace.settings" file: `com.ibm.*=all=enabled`
- Restart the server region for the change to take effect. (Cancel it and WLM will restart it.)
- Output is written to SYSPRINT and/or BUFFERs based on the TRACEBUFFLOC variable in the "current.env" file. Example: `TRACEBUFFLOC=BUFFER SYSPRINT`
  - The default for clients is SYSPRINT, for all others it is BUFFER.
  - The recommended location is SYSPRINT so you can browse it with SDSF. Otherwise, specify BUFFER if you want to use IPCS.
  - For performance reasons, you should not specify SYSPRINT if you don't need it.

- JRAS can also be used for customer applications; see the book "Using the JRas Message Logging and Trace Facilty" in the WAS V4 for z/OS library.

JVM Messaging – This is useful for debugging Java applications.

- Activate by specifying `JVM_DEBUG=1` in the 'current.env' file for the server instance.
  - This is equivalent to the java "`-verbose:class,jni`" option.
- Restart the server region for the change to take effect. (Cancel it and WLM will restart it.)
- Output is written to the server region's SYSPRINT dataset, unless you specify a location in the 'current.env' file such as: `JVM_LOGFILE=/tmp/myjvm.log`

LDAP – There are several useful tools for debugging LDAP and JNDI naming problems.

- LDAP Search command - issue this from the TSO command line:
  ```
  GLDSRCH -h 127.0.0.1 -p 1389 -b "o=<was390_root name>"
  "objectclass=*"
  ```
- LDAP Browser - there are several available; here is one that we like: http://www.iit.edu/~gawojar/ldap
- Activate LDAP tracing with the MVS Modify command for the LDAP server region:
  `f ldap,appl=debug=<nnnnn>`
  - `nnnnn` is a decimal value of the desired debug level. Use '65535' to get everything.
  - To turn the tracing off, use a value of zero: `f ldap,appl=debug=0`
  - Output is written to the LDAP server region's SLAPDOUT data set which you can browse with SDSF.
  - For more information, see "LDAP Server Administration and Usage Guide" - SC24-5861
- For problems with CosNaming registration (i.e., during the Bootstrap), specify `'LDAP_DEBUG=65535'` and `'TRACEBUFFLOC=BUFFER SYSPRINT'` in the current.env file for your naming server to get an LDAP trace.

JDBC – This is useful for diagnosing DB2 SQLJ/JDBC problems.

- Setup the environmental variable parameters in the 'current.env' file:
  ```
  DB2SQLJPROPERTIES=/usr/lpp/db2/db2710/classes/wsccb_db2sqljjdbc.p
  roperties
  ```
- Sample .properties file:
  ```
  DB2SQLJ_TRACE_FILENAME=/tmp/IVP2_jdbctrace
  ```
- Output: JDBC will produce two files:
  - One file will be in binary format in the filename you specified in `DB2SQLJ_TRACE_FILENAME.` Format it using "db2sqljtrace" (see below).
  - The second file contains readable text: `<trace_file>.JTRACE`
- Format the binary trace data using the db2sqljtrace command:

```
db2sqljtrace fmt|flw <input-file-name>
```

- **fmt**: Specifies that the output trace is to contain a record of each time a function is entered or exited before the failure occurs.
- **flw:** Specifies that the output trace is to contain the function flow before the failure occurs.

✍✍ The formated trace output goes to stdout. You may pipe the output to a file.

✍✍ For more information, see "DB2 UDB for OS/390 Application Programming Guide and Reference for Java" - SC26-9932

Component Tracing (CTRACE) - Useful for diagnosing C++ code such as the WAS daemon and other IBM subcomponents (Not for customer applications.)

✍✍ Setup – see the "WAS Installation & Customization" book

- CTRACE options are set in CTIBBOxx in PARMLIB
  `TRACEOPTS WTRSTART(BBOWTR) ON WTR(BBOWTR)`
- Otherwise start it with the MVS operator command: `TRACE CT,WTRSTART=BBOWTR`
- Start tracing for the WAS V4 component:
  `TRACE CT,ON,COMP=SYSBBOSS`
  `REPLY xx,WTR=BBOWTR,END`

✍✍ Output is written to SYSPRINT and/or BUFFERs based on the TRACEBUFFLOC variable in the 'current.env' file:

- Default for client is SYSPRINT, for all others, it is BUFFER
- Specify one or both (SYSPRINT &/or BUFFER)
- Recommend SYSPRINT unless you want to use IPCS.
- For performance reasons, you should not specify SYSPRINT if you don't need it.

✍✍ TRACEALL=x - establishes the general trace level for all WAS components

- (0=None, 1=Exceptions, 2= Basic, 3= Detail)

✍✍ Other Trace Parameters can be specified in the environment file: (See "WAS Messages & Diagnosis" book for details.)

SMF Records - These can be used to capture detailed information about servers, containers, java beans, and methods.

✍✍ You must specify that SMF server and container activity or interval records are to be written for the server definition in the Systems Management Administration tool.

✍✍ Your SMFPRMxx member of PARMLIB must specify that type 120 records are to be written to the SYSx.MANx data sets, and you must extract the records with the IFASMFDP utility to a sequential data set.

✍✍ Detailed information can be displayed with the SMF viewer available from the WebSphere download site at http://www.ibm.com/software/webservers/appserv/download_v4z.html

✍✍ There is an enhanced version of this on http://www.ibm.com/support/techdocs under "White Papers" - see document WP100244.

Other Tools

- There will be a CEEDUMP if there was a failure within the C environment. The "TRACEBACK" section in DUMP is very helpful.
- Distributed Debugger and Object Level Trace – to be documented in another paper.

WSC Sample Programs

- bborblog.sh – a shell script to browse the WAS error log from a telnet session:

```
/* REXX */
/* trace r */
parse arg logstrm format .
if logstrm = '' then logstrm = "WAS.ERROR.LOG"
if format  = '' then format = "80"
qual =userid()
file_name = "/tmp/" || qual || ".errorlog"
"rm " || file_name
"touch " || file_name
call syscalls 'SIGOFF'
call bpxwdyn "alloc fi(bbolog) path('" || file_name || "')"
address LINKMVS "BBORBLOG  logstrm format"
call bpxwdyn "free fi(bbolog)"
"vi " || file_name
exit(0)
```