WebSphere Application Server for z/OS Version 7

APAR OA33516 and _BPXK_DISABLE_SHLIB=YES | NO

Helping avoid OOM conditions with WAS z/OS 31-bit Servers

This document can be found on the web at:
www.ibm.com/support/techdocs
Search for document number WP101320 under the category of "White Papers"

Version Date: August 9, 2010
See "Document Change History" on page 7 for a description of the changes in this version of the document

Written and provided by:
IBM Advanced Technical Skills
WAS z/OS Team ... Gaithersburg, MD
Table of Contents

Summary Overview....................................................................................................................3
SHRLIBRGNSIZE and allocated shared memory in dubbed address spaces..........................3
The value SHRLIBRGNSIZE....................................................................................................3
Example of documented recommendations for large SHRLIBRGNSIZE...............................3
A note to exercise reasonable caution....................................................................................3
What the APAR fix for OA33516 will provide........................................................................4
How it's put into effect with WAS z/OS................................................................................4
Verifying the results................................................................................................................4
Implementing _BPXK_DISABLE_SHLIB for WAS z/OS V6.1 and V7.0..............................4
  Implementing in WebSphere Application Server z/OS V6.1..................................................4
    Controller regions, servant regions and adjunct regions....................................................5
  Isolating the effect to a specific server................................................................................5
Implementing in WebSphere Application Server z/OS V7.0..................................................6
Appendix - Supporting Detail...............................................................................................6
  Verifying the results using IPCS and SVC dump.................................................................6
Document Change History.....................................................................................................7
Summary Overview

**SHRLIBRGNSIZE and allocated shared memory in dubbed address spaces**

The companion WP101320 document, titled "SHRLIBRGNSIZE and the Effect on 31-Bit JVM Storage Needs" spells this out in some detail. In summary:

- **SHRLIBRGNSIZE** is a parameter found in BPXPRMxx. It's used to indicate how much private region to hold aside in dubbed address spaces for shared modules. Shared modules are those marked with the +l extended attribute.

- By default every WAS z/OS region will be dubbed and have the amount of space designated by **SHRLIBRGNSIZE** allocated as shared space at the "top" of the 31-bit addressable private region in subpool 230.

- There are cases where a larger **SHRLIBRGNSIZE** may be of benefit. But it is a system-wide value and applies to all dubbed address spaces, including WAS z/OS.

- The issue is that the bigger the **SHRLIBRGNSIZE** value, the less LE storage available for a JVM to acquire heap. For JVMs operating in 31-bit mode, this can be a problem.

- You may wish *not* to allocate that space in some address spaces. That was not possible in the past. With the APAR fix to OA33516 it will be.

**The value **SHRLIBRGNSIZE**

We do not wish to imply **SHRLIBRGNSIZE** is itself a problem. Sharing common modules across address spaces reduces the amount of real storage used, particularly when many regions use the same modules. Less real storage consumed may translate to less paging, depending on how tight you are on real storage.

**Example of documented recommendations for large **SHRLIBRGNSIZE**

For a 1.6GB private region size a WAS z/OS server will get to use about 1.2GB. From that 1.2GB will be subtracted whatever the **SHRLIBRGNSIZE** value is.

The IBM Lotus Domino z/OS product documentation recommends a **SHRLIBRGNSIZE** setting of 500MB. That leaves 700MB. Some portion of that will go to the JVM native libraries, leaving something less than 700MB.

An attempt to acquire JVM heap beyond what's available will result in an out of memory condition.

Prior to APAR fix OA33516 your options were to reduce the 500MB **SHRLIBRGNSIZE** value (and incur whatever effect that has on your Domino environment), or live within the imposed heap limitations. Depending on your Java application needs that may not be possible.

Another vendor's relational database recommends 350MB for **SHRLIBRGNSIZE**. A smaller number than Lotus Domino, but the general effect is the same.

What the APAR fix for OA33516 will provide is the ability to define a large **SHRLIBRGNSIZE** value for the LPAR but disable the effect on individual WAS z/OS server regions.

**A note to exercise reasonable caution**

The APAR fix for OA33516 should be used for those regions where out of memory conditions are experienced for 31-bit JVMs.

*By no means do we wish to imply all WAS z/OS regions should disable the use of shared memory with the APAR fix for OA33516. On the contrary, we advise you use caution when disabling shared memory, particularly if you run tight on real storage to begin with.*

---

1 In this sense it's like LPA.
**What the APAR fix for OA33516 will provide**

Note: APAR OA33516 is designated for z/OS 1.10 and higher and is due out later in 2010. Watch for it if you have 31-bit WAS z/OS out of memory problems. In RETAIN it is "New Function".

It will introduce a new `_BPXK_DISABLE_SHLIB=YES|NO` environmental property to control whether or not a given address space will have the shared library region mapped into the address space. The effect is:

- If **NO** (the default) the amount of space specified by `SHRLIBRGNSIZE` will be carved out of each WAS z/OS server address space. This is what happens today without the new environmental property. If **NO** or unspecified, then the behavior is unchanged from what was spelled out in the companion "SHRLIBRGNSIZE and the Effect on 31-Bit JVM Storage Needs" document.

- If **YES** then modules will be loaded into the private area as if they were not in the shared library region. **No shared region will be carved out of the address space**. This can save significant amounts of virtual storage in each WAS z/OS address space when `SHRLIBRGNSIZE(nnn)` is the default (64MB) or larger.

**How it's put into effect with WAS z/OS**

This depends on the version of WAS z/OS:

- For WAS V6.1 it takes the form of a `//CEEOPTS DD` statement in the JCL start procedures for the servers. We describe this under "Implementing in WebSphere Application Server z/OS V6.1" starting on page 4.

- For WAS V7.0 it takes the form of a normal WAS environment variable scoped to your needs. This is described under "Implementing in WebSphere Application Server z/OS V7.0" starting on page 6.

**Verifying the results**

There are two methods to accomplish this:

1. Apply the APAR fix for OA33516 and enable `_BPXK_DISABLE_SHLIB=YES`. If your out of memory problems go away, then it's a good indicator the desired result has taken effect.
2. Use a tool that will map virtual storage in an address space. We illustrate this under "Verifying the results using IPCS and SVC dump" starting on page 6. This would work for WAS z/OS V6.1 or Version 7.

**Implementing `_BPXK_DISABLE_SHLIB` for WAS z/OS V6.1 and V7.0**

Note: If you're not experiencing any out of memory issues then you may not need to consider using this new function. However, you may wish to explore its use for some anticipated future need.

**Implementing in WebSphere Application Server z/OS V6.1**

For WAS z/OS V6.1 this new function is enabled with a `//CEEOPTS DD` statement in the server JCL start procedure that points to a data set member with the `_BPXK_DISABLE_SHLIB` parameter and its value:
That raises a couple of questions:

1. What region types should this be applied to -- controller, servant and/or adjuncts?
   
   **Hint:** any or all, but the servant region is the one where OOM issues typically arise

2. How can we isolate the function to a one server rather than all in the node?
   
   **Hint:** by creating a separate JCL proc and include member and changing the server’s start properties

**Controller regions, servant regions and adjunct regions**

The \_BPXK\_DISABLE\_SHLIB function applies to all three types of regions -- controllers, servants and adjuncts. It applies to application servers, Node Agents, Deployment Managers or Daemon servers.

But it’s the servant regions that host the user applications, and it’s the servant regions that most often experience the out of memory issues.

**Key Point:** While you can use this function with any or all of the region types, applying it to the **servant region** provides the greatest benefit.

**Isolating the effect to a specific server**

By default there is one controller, servant and adjunct JCL start procedure (and an include member) for a node. By design the same procs may be used by all the servers in a node. The server to start is specified by passing in the **ENV**= parameter on the **START** command.

That means by default adding \_BPXK\_DISABLE\_SHLIB=**YES** affects all the servers in the node. That may be exactly what you want. If so, you need not do anything more.

But you may wish to have this function apply only to a single server experiencing out of memory issues. To accomplish this you must make a copy of the JCL proc members and tell WAS to use the different procs for that server.

**Making copies of JCL start procedure members**

Determine the region type (or types) you wish to isolate for purposes of this new function and make a copy of the appropriate members.

Be sure to copy the main body member as well as the referenced include member (often referred to as the "Z-member")

Use whatever naming convention is appropriate. We recommend you maintain the same two character cell identifier at a minimum.

Add the \/_CEE0PTS DD statement with the pointer to the dataset and member containing the \_BPXK\_DISABLE\_SHLIB=**YES** property.\(^2\)

---

\(^2\) New controller procs may require a new **STARTED** profile, depending on how discrete the current profile is. Servant and adjunct regions are started with **ASCRE** and **STARTED** profile is based on the server short, which isn’t changed.
**Updating WAS configuration to indicate use of new start procedures**

This is accomplished through the Administrative Console by drilling down to the server's configuration, then drilling further as shown. Add the new JCL proclib member names:

![Diagram of WAS configuration](image)

Save and synchronize, then restart the server.

**Implementing in WebSphere Application Server z/OS V7.0**

This is easier in that the `_BPXK_DISABLE_SHLIB=YES` is set as a WAS environment variable scoped to whatever level is appropriate: cell, node, cluster or server. There is no need to make copies of JCL proc members or add DD cards.

**Note:** Unlike with V6.1, this will apply to CR, SR and Adjuncts. There's no way to isolate to a specific region type. That's okay ... as mentioned, this works for all region types.

The variable is created like any other WAS variable is created:

![Variable creation](image)

When the variable is saved and synchronization takes place, the value is propagated out to the `was.env` file of the server (or servers if scope cluster, node or cell). `was.env` is what's pointed to by `//STDENV DD` in the JCL proc.

---

**Appendix - Supporting Detail**

**Verifying the results using IPCS and SVC dump**

To verify that the function is taking effect you may use a tool that will map virtual storage in an address space.

*Before* you make the change, you will find an allocated block of storage the size of your system's `SHRLIBRGNSIZ(xxx)` value in SP 230 Key 0 storage.

*After* you have made the change, it will be absent from the virtual storage map.

One tool we suggest is IPCS to look at an SVC dump. For example, on our system the `SHRLIBRGNSIZ(xxx)` value was 64MB, which equates to `0x4000000`.

We invoked IPCS on the dump, went to the component screen (`Main ⇄ Analysis ⇄ Component`) and invoked `VSMDATA`. Then we searched for string:
***** Subpool  230, key 0 Total alloc:
until we found one that had a length larger than our '0x4000000' shared library region size.

**Note:** There may be more than one Subpool 230 Key 0 area of storage larger than the shared library region size. The objective was to narrow the search, then match back to the DQE that exactly matched the shared library region size of our system.

We saw this Subpool 230 Key 0 allocation, which looked promising:

***** Subpool  230, key 0 Total alloc: 43B1000 (1000 Below, 43B0000 Above)

We then paged toward the top to look at the DQEs and find the DQE that matched the size of the region. We saw this:

DQE at address 7F66CF58
NEXT..... 7FF177F0  PREV..... 7F66C850  FFQE..... 7F66C850  LFQE..... 7F66C850
AREA..... 008CD000  SIZE..... 00001000

FQE at address 7F66C850
AREA..... 008CD000  SIZE..... 0000F80  NEXT..... 7F66CF58  PREV..... 7F66CF58
DQE...... 7F66CF58

Descriptor Queue Element queue (Virtual 31, Real 31) data follows:

DQE at address 7F6001C0
NEXT..... 7E8B3820  PREV..... 7F600148  FFQE..... 7F6001C0  LFQE..... 7F6001C0
AREA..... 78E00000  SIZE..... 04000000

DQE at address 7E8B3820
NEXT..... 7E8B3820  PREV..... 7F6001C0  FFQE..... 7E8B3820  LFQE..... 7E8B3820
AREA..... 7E86C000  SIZE..... 00005000

The DQE at address '0x7F6001C0' had a size of '0x04000000' which matched exactly the SHRLIBRGNSIZ value of our system.

**Important!** The presence of that DQE that exactly matched our SHRLIBRGNSIZ value indicates we were not using the new _BPXK_DISABLE_SHLIB=YES function.

The absence of a DQE that exactly matched the SHRLIBRGNSIZ value of the system would indicate that the function is in place and in effect.

---

**Document Change History**

Check the date in the footer of the document for the version of the document.

**Aug 9, 2010** Original document.

---

End of WP101320