Servlet Filter for all Web Applications
Plus writing user data to SMF
for WAS V7 on z/OS

WP101859

Version 1.3

Author: Edward McCarthy
IBM
edwardmc@au1.ibm.com

Last Updated: 07 February 2011, 18:44

Table of Contents

1 Servlet filter for all web applications

1.1 Introduction

In this techdoc we provide sample code that demonstrates two capabilities of WebSphere Application Server V7 on z/OS:

1. enabling a servlet filter for all deployed applications without having to modify them
2. demonstrate how to add user data to the SMF 120 subtype 9 records

1.1.1 Acknowledgements

The author would like to acknowledge the assistance of Mike Cox of the Washington Systems Center who developed the key code that provided the capability to create servlet filters for deployed web applications.

1.1.2 Servlet filter for all deployed applications

The J2EE specification provides a way to configure a servlet filter for a web application. Typically this involves appropriate configuration of the web.xml file contained in the ear file. This approach is described here:

http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.zseries.doc/info/zseries/ae/cweb_sfilt.html

This document and the supplied zip file describe an approach which allows you to enable a servlet filter for all web applications deployed in a WAS V7 server, without requiring any modification of the web.xml file of the application ear file.

This approach means that no changes are required to already deployed applications.

1.1.3 Use on non z/OS environments

The target WAS environment used in this document is z/OS. However the approach described in this document and the code supplied with this document will work with WAS V7 on any support operating system such as Windows, AIX etc.

The SMF API part in the sample servlet filter is not executed on the distributed environments since they do not have the z/OS SMF capability.

1.1.4 Adding user data to SMF record

WebSphere Application Server V7 on z/OS provides the capability to create a SMF 120 subtype 9 record for each request processed. Details about this can be found here:

http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.zseries.doc/info/zseries/ae/rtrb_SMFsubtype9.html

A feature of this capability allows applications to write their own user data to the SMF record.

To demonstrate how to write the code to do this, we decided to add this code as part of the servlet filter example. In the servlet filter we check the request to see if it has a query string and if it does we write the query string as user data to the SMF record.

The default behavior is to write the user data with a type value of 65999.

1.1.5 Supplied files

A zip file is supplied called: techdoc-WP101859-webContainer-servletFilter.zip

This file contains the following:

File	Description
webContainerFilter.jar	jar file containing the servlet filter
was-manage-jvm-customProps.jy	program to help define custom props
wsc-wsadmin-jvm-cmds-manage-custProps.txt	sample wsadmin commands
ibm-wsc-webContainer-servletFilter.zip	RAD project interchange file, contains the source code
smfDumpJcl.txt	JCL to dump SMF 120 records to a dataset
idcamsJcl.txt	JCL to run IDCAMS to dump SMF records
smfPrinted120Records.txt	Output from running bbomsmfv.jar
smfDump02.txt	Output from running IDCAMS

1.2 Why the capability to add servlet filter without updating web.xml

The J2EE specification provides for the servlet filter capability, which is implemented in WebSphere Application Server. A servlet filter allows you to perform some action of your choice such as writing some logging data.

The standard approach to implement a servlet filter is to modify the web.xml file that is packaged as part of an ear file containing the web application.

A drawback with this approach is that customers may want to implement a common servlet filter to all or most of their web applications to provide some common functionality. This then means that application developers have to update their web.xml file accordingly.

Consider for example a company that has a large number of applications already deployed in WebSphere Application Server. Imagine that a new business requirement requires that all these applications implement a common servlet filter. The normal approach would require modifying the web.xml file of each deployed application. This would require testing of each application and take considerable time.

A preferable approach would be to be able to enable a servlet filter for all the web applications without application developers having to be concerned with updating the web.xml file.

That is what the sample jar file provided with this techdoc provides. Using the supplied jar file you can enable a servlet filter for your web applications without having to update them in any way.

1.2.1 Overview of how this is achieved

To implement this capability requires developing two java classes. One class implements the ServletContextListener interface, the other implements the Filter interface.

The class that implements the Filter interface is coded as any normal servlet filter interface.

The class that implements the ServletContextListener interface performs the process of creating the servlet filter class for each web application in the WAS server.

1.3 Servlet filter and adding user data to SMF

Having developed the code to enable creating a servlet filter for deployed web applications, we wanted to show the filter performing some sort of action.

A customer query resulted in deciding to show how to add user data to the SMF 120 subtype 9 records.

The customers query related to how to save the query string part of a URL to the associated SMF 120 subtype 9 records.

A request with a query string looks similar to this:

http://www.someCompany.com/doUpdate?account=123&deposit=100

In the above request everything following the '?' is the query string.

The SMF 120 subtype 9 record written for this request only contains the contents of the URL up to the '?'. The customer in this case wanted to have the query string data saved to the SMF record as well.

WebSphere Application Server V7 provides an API that allows application programs to add user data to the SMF record for each request.

As the customer wanted to do this for all their deployed applications, this tied in nicely with the method for implementing a servlet filter for all deployed web applications without having to modify them.

1.4 Servlet filter processing sequence considerations

The processing of servlet filters that are configured the standard way, that is by making the appropriate changes to the web.xml file in application ear files, is not affected by use of the servlet filter mechanism described in this document,

If you set up the use of the servlet filter mechanism described in this document in a server, with applications that already have servlet filters associated with them, then the order of processing is:

1. Servlet filters set up using the mechanism described in this document are processed first
2. Processing of servlet filters set up in the web.xml of applications ear files

occurs next

The only consideration to keep in mind is if the servlet filter implemented by the means described in this document preformed some action that did not allow the normal flow of processing within WebSphere Application Server to continue.

In the servlet filter code is this line:

```
chain.doFilter(request, response);
```

If you created your own servlet filter and this line was not executed then normal processing of the servlet does not continue.

1.4.1 Can I have multiple servlet filters?

The sample code described in this document shows how to implement a servlet filter for all or some number of applications deployed in a WAS server.

Later in this document in the section headed 'Define listeners property', you will see how you can specify the "listeners" property on the server to specify the name of the class that provides this capability.

The class you specify can only add one servlet filter to an application.

If you had a requirement to add a number of servlet filers to deployed applications then you can achieve this.

This link from the WebSphere Application Server V7 infocenter describes the "listeners" property:

http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.nd.iseries.doc/info/iseriesnd/ae/rweb_custom_props.html

At this link it describes how you can specify multiple class names:

> The value for this property is a string specifying a comma separated list of listener classes. The listener supplied must implement standard listener classes from the Java Servlet API or IBM® listener extension classes.

If you do specify multiple classes, then each class can add one servlet filter.

The servlet filters are executed in the order corresponding to the order of the classes specified on the "listeners" property. We tested using two classes in the "listeners" property to verify this behavior.

1.5 How to implement the sample

The following describes how to set up your WAS V7 environment to use the supplied jar file.

The default behavior of the supplied jar file is to enable a servlet filter for all deployed web applications in the server. The following describes the few steps that need to be done to achieve this.

Following that we will describe some additional properties you can define should you only want to enable the servlet filter for some web applications in the server.

1.5.1 Copy jar file to classes directory

A jar file called webContainerFilter.jar is supplied with this document. It must be placed into the classes directory on the node where the WAS server is running.

To confirm which directory, check in the WAS server log, look for 'WAS_INSTALL_ROOT '. You should find a line similar to the following:

```
BBOJ0077I:     ${WAS_INSTALL_ROOT} =
/WebSphereEd/wycell/node55
```

This implies that the directory you need to copy the jar file to is:

```
/WebSphereEd/wycell/node55/AppServer/classes
```

Note when copying the file from a Windows machine to this directory on z/OS, use the binary transfer option.

After copying the file to this directory, check the permissions and owner etc to make sure the userid that the WAS servant started task is running under has read access to the file. The file only requires read permission to be set.

Note if you have your cell running over multiple LPARs, then other servers on other nodes would be using the classes directory under that node, and the jar file must be copied there as well.

Note when copying the file from a Windows machine to z/OS, use the binary transfer option.

1.5.1.1 *Classes directory on distributed environments*

On distributed systems, you may need to create the classes sub-directory. We tested the code using WAS V7 on a Windows system and found we had to create the classes sub-directory in this location:

C:\zProducts\IBM\WID7_WTE\runtimes\bi_v7\profiles\qwps\classes

As for WAS V7 on z/OS, the classes directory needs to be defined under the profile directory.

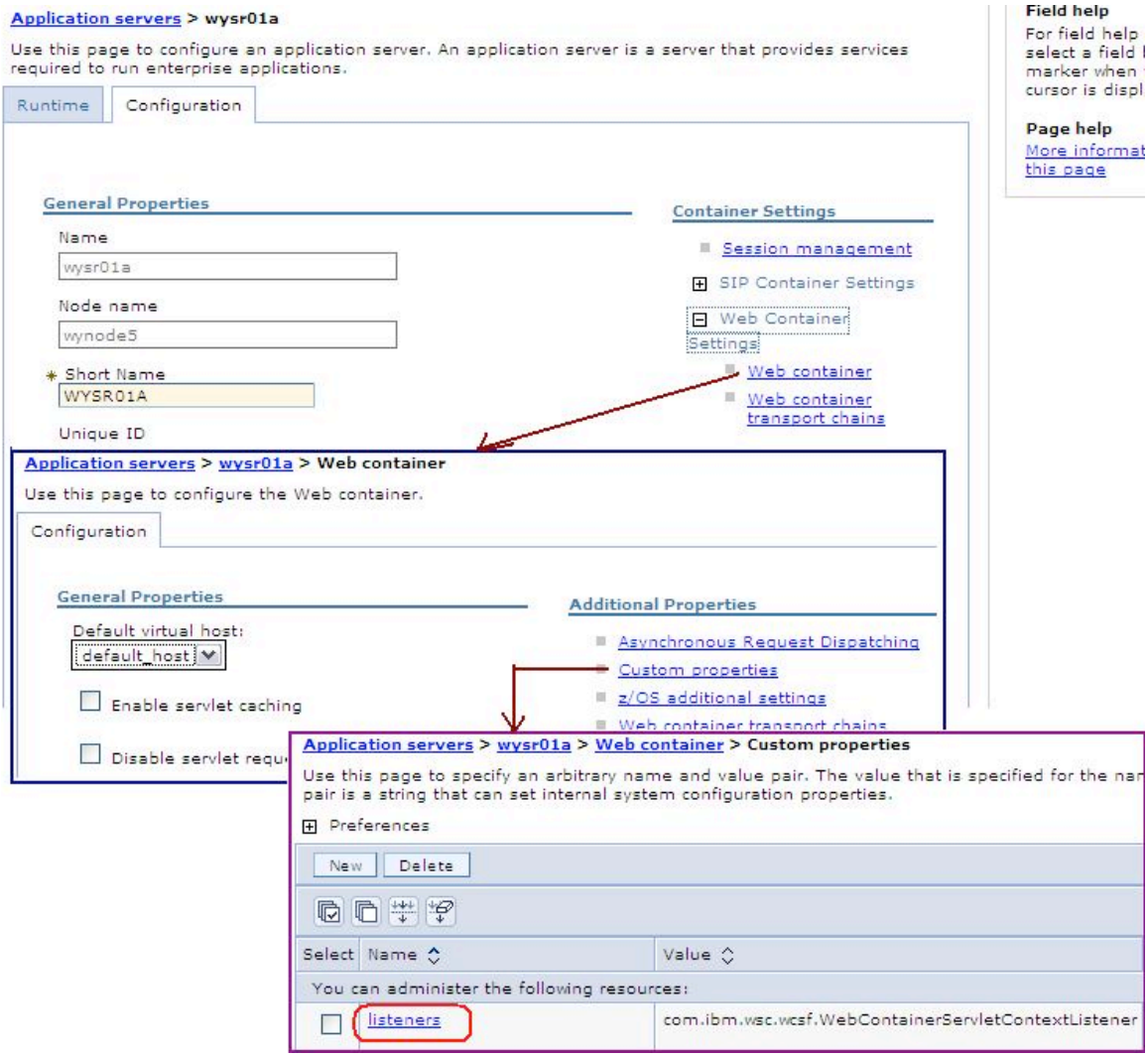
1.5.2 Define listeners property

Using the WAS V7 administration GUI, define a custom property called 'listeners' with a value of: com.ibm.wsc.wcsf.WebContainerServletContextListener

To do this select the server, then Web Container Settings -> Web Container, then Custom Properties and then click the New button and define the property. Save the change to the WAS master configuration and sync the change to the node.

You need to do this for each WAS server that you want to implement this capability in.

Below shows a compilation of screen shots on how to do this:



1.5.2.1 *Detecting if class not loaded successfully*

If you specify an invalid class name or the WAS server is unable to load the class then in the WAS servant log you will see a message similar to this, which can be found by searching for the string loadListener:

```
Trace: 2011/02/07 10:35:59.665 01 t=7C77A0 c=UNK key=P8
(13007002)

  ThreadId: 00000006

  FunctionName:
com.ibm.ws.ffdc.impl.FfdcProvider.logIncident

  SourceId: com.ibm.ws.ffdc.impl.FfdcProvider

  Category: INFO
```

```
  ExtendedMessage: FFDC1003I: FFDC Incident emitted on
/WebSphereEd/wycell/node55/AppServer/profiles/default/logs/f
fdc/wycell_wynode5_wysr09a_WYSR09AS_STC26342_000003380000001
A_43884388_11.02.07_10.35.59.5858417581557586232002.txt
com.ibm.ws.webcontainer.srt.WebGroup.loadListener
```

If you binary transfer this FFDC file to a Windows PC and edit it then you will find a message similar to this:

```
[2/7/11 10:35:59:593 GMT]     FFDC
Exception:java.lang.ClassNotFoundException
SourceId:com.ibm.ws.webcontainer.srt.WebGroup.loadListener
ProbeId:1527
Reporter:com.ibm.ws.webcontainer.WSWebContainer@27142714
java.lang.ClassNotFoundException: class
java.lang.ClassNotFoundException:
com.ibm.wsc.wcsf.WrongNameWebContainerServletContextListener
      at java.beans.Beans.instantiate(Beans.java:194)
      at java.beans.Beans.instantiate(Beans.java:75)
      at
com.ibm.ws.webcontainer.WebContainer.loadListener(WebContain
er.java:378)
```

The above shows that the WAS server generated a class not found exception when it tried to load the specified class.

1.5.3 Test setup

Having copied the jar file to the appropriate directory and defined the listeners property and restarted the server you can start testing immediately.

The default behavior is that a servlet filter will be associated with each deployed web application in the server with the filter setup to process each request.

You will see this message in the WAS servant log confirming that a servlet filter will be established for each web application:

```
>>> Servlet filters will be created for each web application
```

To test, simply run a request that contains a query string from one of your applications.

Your request should run as normal.

No messages indicating the servlet filter has been invoked etc are written. Later in this document we will describe a debug property that can be set to have debug messages written.

1.5.4 Verifying user data added to SMF record

To verify that the query string has been added to the SMF user data, the simplest way is to print the SMF record and scan the output to verify that the presence of the query string.

On our z/OS system we used the following process to achieve this. Your own z/OS environment may provide alternative methods to achieve the same thing.

1.5.4.1 *Switch SMF datasets*

We issued the command I SMF to switch SMF recording to the next SMF dump data set, so that we only had a few SMF records to process.

1.5.4.2 *Run request with query string*

We then ran this request:

http://wtsc55.itso.ibm.com:7127/IBMTools/EBizSuperSnoop?AAAAA=1111111

1.5.4.3 *Extract SMF 120 records from SMF*

We then used the JCL in the file smfDumpJcl.txt to extract the SMF 120 records from the current SMF dataset to a dataset. We checked in the job output that the column titled 'RECORDS WRITTEN' had a non zero value for 120 records.

1.5.4.4 *Print SMF record using bbomsmfv.jar*

There is a freely available java utility called bbomsmfv.jar that can be used to format the SMF 120 records. Details about how to obtain and use this utility can be found here:

http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.zseries.doc/info/zseries/ae/ttrb_SMFviewdata.html

From a telnet session to the z/OS LPAR we ran the utility against the dataset containing the extracted SMF 120 records using this command:

```
java -cp bbomsmfv.jar com.ibm.ws390.sm.smfview.SMF
"INFILE(EDMCAR.SMFDATA.FEB02A)"
"PLUGIN(DEFAULT,/u/edmcar/smfDumpFeb02.txt)"
```

In the output file we found an SMF record containing this part showing the URI of the request:

```
Triplet #: 7; Type: ClassificationDataSection;
  Version     : 1;
  Data Type   : 6 (URI);
  Data Length: 24;
  Data        : /IBMTools/EBizSuperSnoop (EBCDIC);
```

And this part showing the user data section containing the query string:

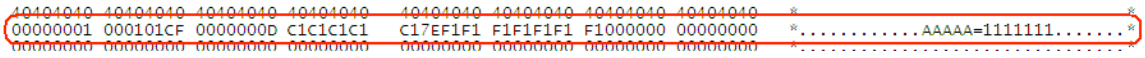
```
Triplet #: 9; Type: UserDataSection;
  Version     : 1;
  Data Type   : 65565;
  Data Length : 13;
  Data                  * c1c1c1c1 c17ef1f1 f1f1f1f1 f1000000 *
                        * 00000000 00000000 00000000 00000000 *
```

Note the user data is only printed in hexadecimal format, but the hex string '`c1c1c1c1 c17ef1f1 f1f1f1f1 f1000000`' is the query string 'AAAAA=1111111'. The supplied file smfDetailedReport.txt contains the above output.

1.5.4.5 *Print SMF record using IDCAMS*

We also used the JCL in the file idcamsJcl.txt to print the extracted SMF 120 records. The JCL executes IDCAMS to dump the SMF record content.

Looking through the output we found this:



In the above display you will see the hex bytes 000101CF, this converted to decimal is the value 65999, which is the default value used by the filter.

The next four bytes '0000000D' is the length in hexadecimal of the user data.

Following that is the query string.

The supplied file smfIdcamdsSmfDump.txt contains a sample output with the above output.

Note a shortcoming of IDCAMS is that it does not print lower case characters, so these do not show up on the right hand side of the output.

1.5.5 Servlet Filter property for SMF user data

The servlet filter implemented in this sample shows how to write user data to the SMF 120 subtype 9 record. When the user data is written, one of the parameters for the API is a number.

The servlet filter code uses a default value for this number of 65999, but you can override this by setting the following property on the WAS JVM:

wsc.servletFilter.smfUserDataKey

This property specifies a number, it must be greater the 65535.

The number is used in the addDataToSMF120Subtype9Record API to help identify the user data added to the SMF 120 subtype 9 record

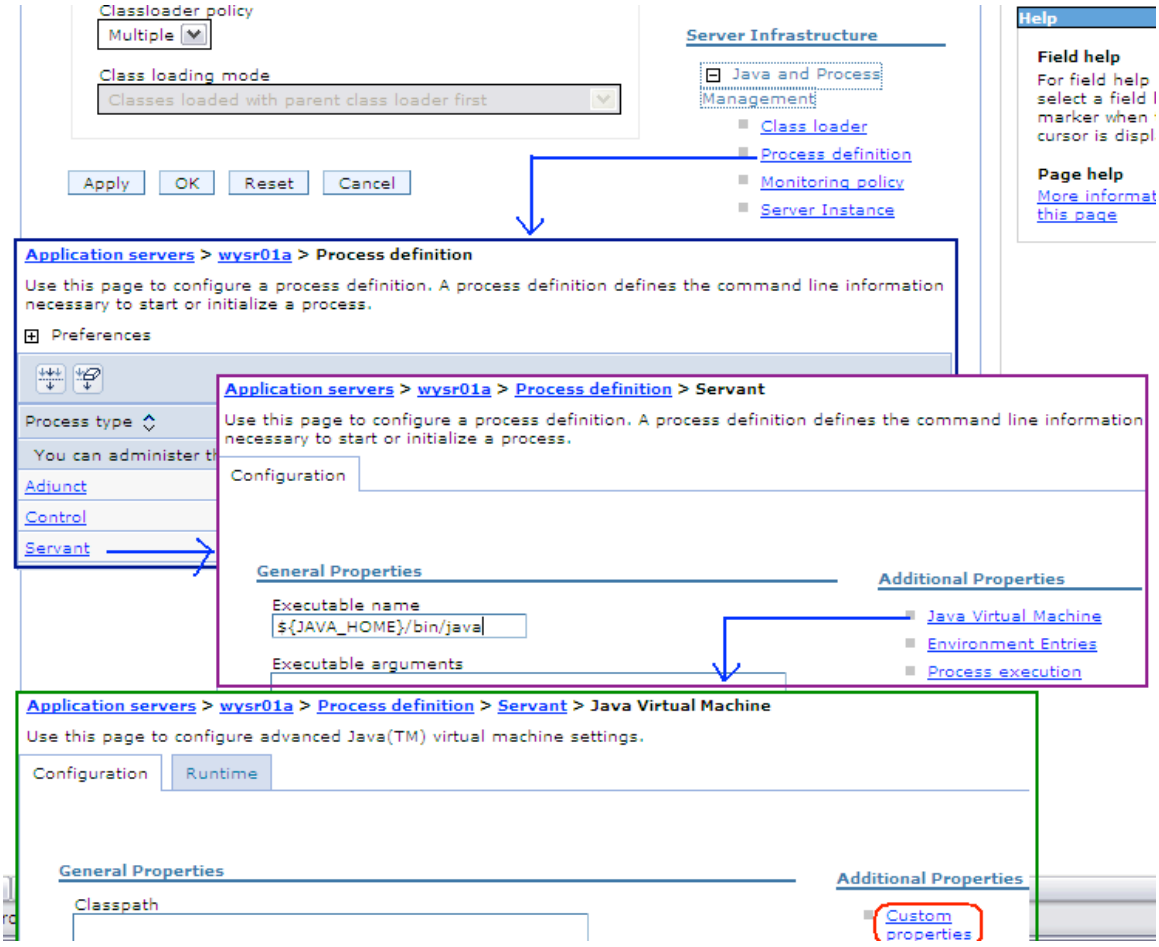
If this property is not defined a default value of 65999 is used.

This property should only be defined once per WAS server.

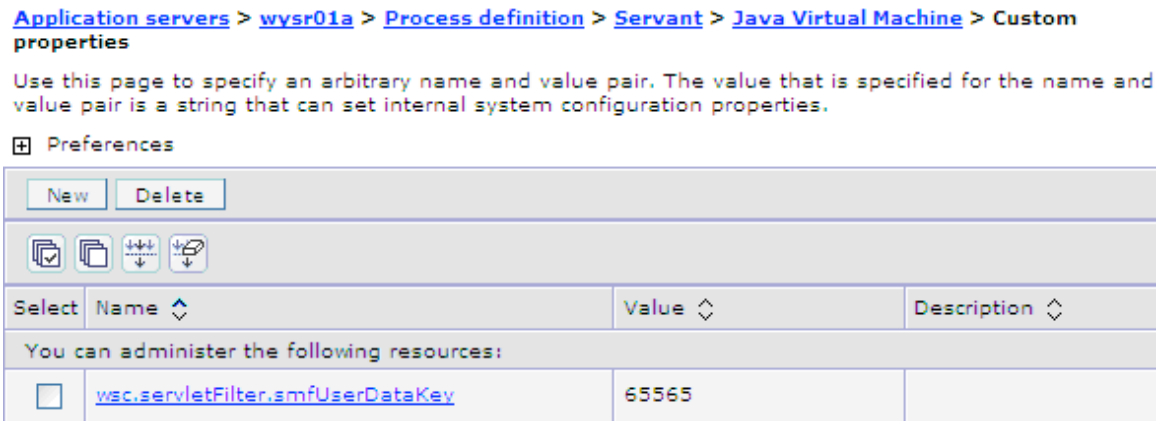
Use the WAS V7 administration GUI to define the above JVM custom property.

To do this select the server, then under the Server Infrastructure heading expand Java and Process Definition, then click Process Definitions, then on the next screen click Servant, then on the next screen click Java Virtual Machine, then on the next screen click Custom Properties. On that screen use the New button to define the required custom properties. Save and sync your changes when done. A restart of the WAS server will be required to bring these changes into effect.

Below shows a compilation of screen shots on how to do this:



Below shows the defined custom property:



Note that any property you define in this manner also show up in the WAS servant job log, for example:

```
BBOJ0077I: wsc.servletFilter.smfUserDataKey = 65565
```

1.5.6 Properties to limit Servlet Filter activation

If you only want to have the servlet filter activated for some of the deployed web applications, then you can use the property called wsc.servletFilter.contextRoot.n to achieve that.

Further if you only want the servlet filter to act on certain requests then you use the property called wsc.servletFilter.contextRoot.n.mappingFilter.x to achieve that.

1.5.6.1 *Using the wsc.servletFilter.contextRoot.n property*

The property called wsc.servletFilter.contextRoot.n is used to specify the context roots of the web applications you want a servlet filter enabled for.

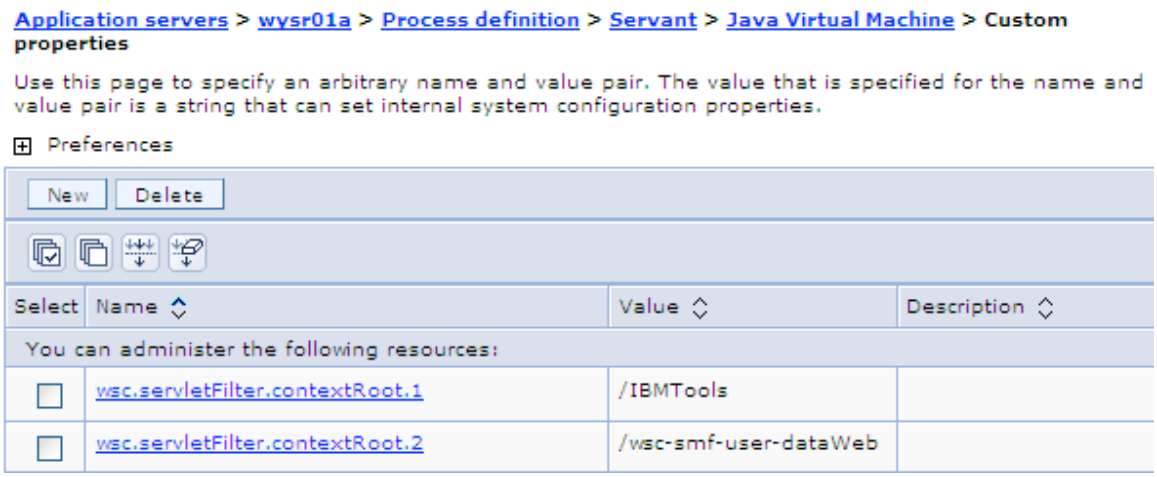
The n part of the property name is a number. The first such property must start with n set to 1. Each subsequent property must have n increased by 1. Any number of these properties can be defined.

The value of the property is the context root on a deployed web application and must include a forward slash, that is a /.

If you use this property, you will see this message in the WAS servant log:

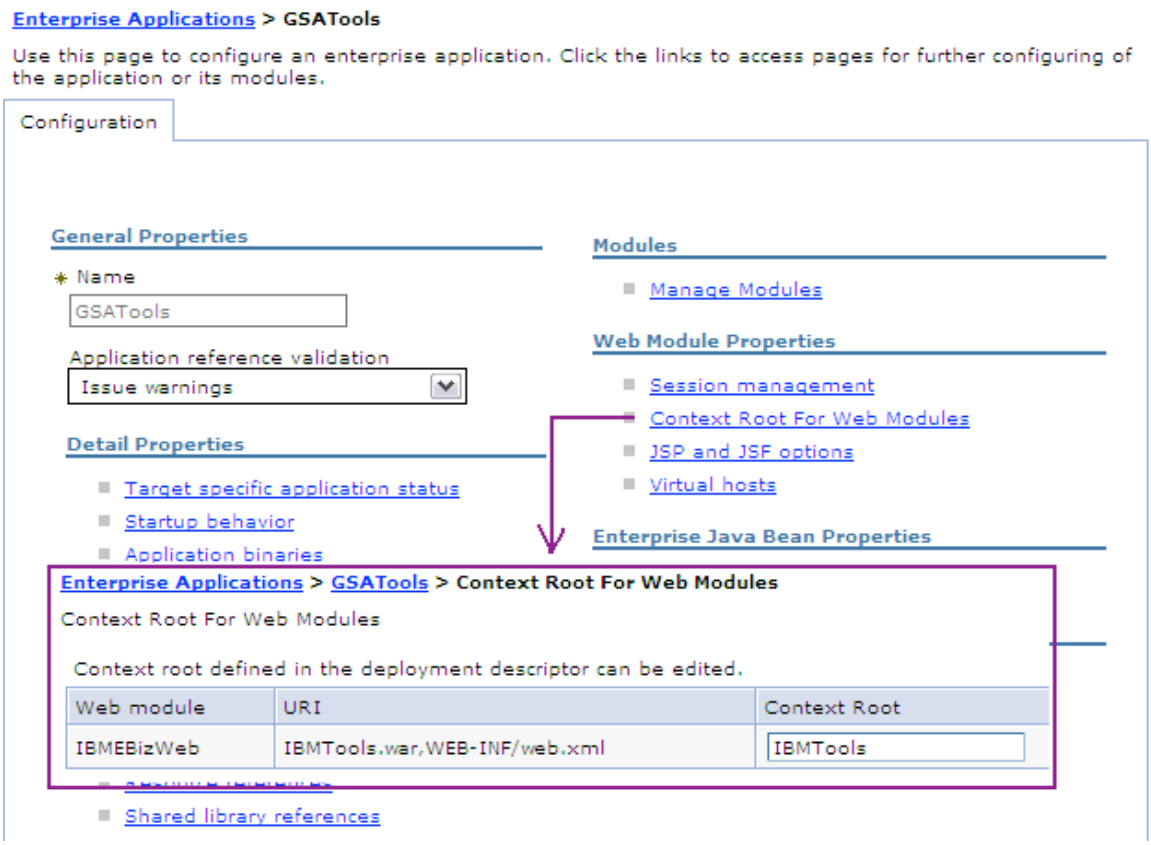
```
>>> Servlet filters will be created for selected web
application(s)
```

In our testing we defined two such properties as shown in the following screen shot from the WAS V7 administration GUI



If you have a web application for which the context root is just /, then you would set the property to the value /.

To locate the context root for deployed applications you can use the WAS V7 administration GUI as shown below:



Note that in the display above, the context root does not show a leading /.

When defining the wsc.servletFilter.contextRoot.n property, be sure to code a leading /.

1.5.6.2 *Using the wsc.servletFilter.contextRoot.n.mappingFilter.x*

The property called wsc.servletFilter.contextRoot.n.mappingFilter.x provides a way to specify what requests the servlet filter is to act upon. The code uses the values supplied via this property to set mapping filters on the servlet filter. The result is that the servlet filter is only invoked by WebSphere for requests that match the mapping filters.

The values you can set for this property follow the same rules that are used when you define mapping filters in web.xml files.

Note you do not need to define this property at all. If you do not define it then the default behaviour is that the servlet filter will act on all requests.

Any number of wsc.servletFilter.contextRoot.n.mappingFilter.x properties can be defined for a corresponding wsc.servletFilter.contextRoot.n property.

The n part of the property name corresponds to the n part of a correspondingly defined wsc.servletFilter.contextRoot.n property.

The x part of the property is a number that must start from 1 with values increasing by 1.

Some examples should help to clarify how to code this property.

Assume you have defined these two properties for two web applications:

```
wsc.servletFilter.contextRoot.1=/IBMTools
```

```
wsc.servletFilter.contextRoot.2=/testApp
```

Then for the above web applications, to control what requests the servlet filter is invoked for you would define the following properties:

```
wsc.servletFilter.contextRoot.1.mappingFilter.1=jdbcConnPool
wsc.servletFilter.contextRoot.1.mappingFilter.2= EBizSuperSnoop
wsc.servletFilter.contextRoot.2.mappingFilter.1=updates
wsc.servletFilter.contextRoot.2.mappingFilter.2=queries
wsc.servletFilter.contextRoot.2.mappingFilter.3=*.jsp
```

In the above set of properties, the first two set mapping filter values for the web application with a context root of /IBMTools, while the last three properties set mapping filter values for the web application with a context root of /testApp.

Details about what sort of values you can code for the mapping filter can be found in the Java™ Servlet Specification Version 2.5, in chapter 11, which can be downloaded from here:

http://winstone.googlecode.com/files/servlet-2_5-mrel2-spec.pdf

Note that the use of masking in the supplied values is limited. For example coding this:

```
wsc.servletFilter.contextRoot.1.mappingFilter.1=jdbc*
```

to imply that you wanted the filter servlet to only process requests starting with jdbc would not work, as jdbc* is not a valid value that can be set as a mapping filter value. This is a restriction of the J2EE specification.

1.5.7 Supplied helper wsadmin program

Defining the custom properties via the WAS V7 administration GUI can be a time consuming task if you have to do this process for several servers.

Supplied is a jython program called was-manage-jvm-customProps.jy. You can use this program to define the properties you require from a command line on your z/OS LPAR.

Note when copying the file from a Windows machine to z/OS, use the binary transfer option.

The file can be copied to any suitable directory on your z/OS system.

Also supplied is a file called wsc-wsadmin-jvm-cmds-manage-custProps.txt. This contains sample wsadmin commands that show how to use the jyhton file to add, update, delete and display all the WAS server JVM custom properties.

The file contains comments about the parameters required etc.

1.5.8 Debugging

The standard java logging mechanism has been used in the supplied code. If you have problems with implementing this supplied sample then you can enable detailed debugging messages by setting trace strings in the WAS server.

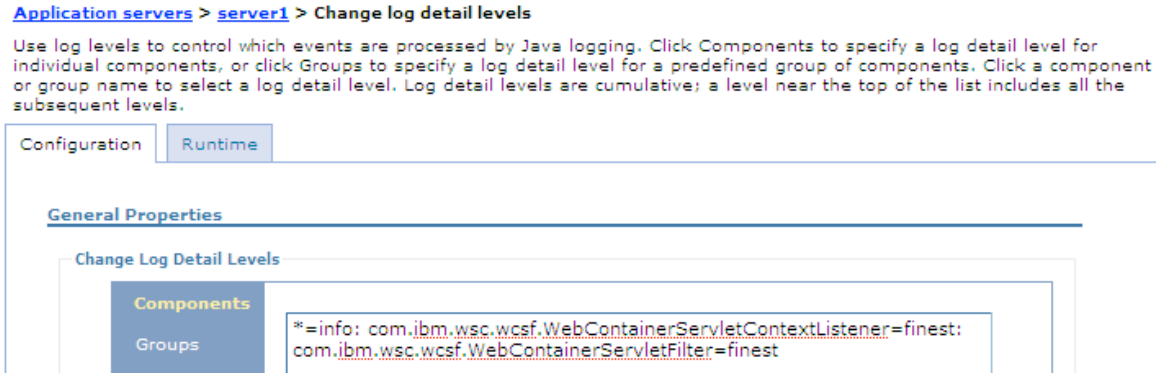
To trace the class `com.ibm.wsc.wcsf.WebContainerServletContextListener` use this trace string:

```
com.ibm.wsc.wcsf.WebContainerServletContextListener=finest
```

To trace the class `com.ibm.wsc.wcsf.WebContainerServletFilter` use this trace string:

```
com.ibm.wsc.wcsf.WebContainerServletFilter=finest
```

The following screen show shows where in the WAS server you can set these trace entries:



Note all processing by the `com.ibm.wsc.wcsf.WebContainerServletContextListener` class occurs during WAS server startup, setting tracing on for this class dynamically after a restart would not produce any trace messages.

The servlet filter class `com.ibm.wsc.wcsf.WebContainerServletFilter` is called during normal processing so you could dynamically enable tracing for this class to obtain debug messages if required.

If you enable tracing the trace entries in the WAS servant log will be similar to this:

```
Trace: 2011/02/07 07:46:24.573 01 t=7C5470 c=UNK key=P8
(13007002)
  ThreadId: 00000022
  FunctionName:
com.ibm.wsc.wcsf.WebContainerServletFilter.doFilter
  SourceId: com.ibm.wsc.wcsf.WebContainerServletContextListener
  Category: FINEST
  ExtendedMessage:  SMF User data key: 65565 SMF Add user data
rc: 0 smfRcDesc: Added data successfully
```

2 Information about the servlet filter program

In this chapter we will discuss various aspects relating to the code supplied with this techdoc.

2.1 The server filter code

There are two classes to provide the servlet filter functionality described in this document.

The class WebContainerServletContextListener is the class that controls setting up the servlet filters for the web applications.

The class WebContainerServletFilter contains the code that will be executed for each request that matches the associated mapping filter.

At the start of the class is a static area which is used to check if any properties described previously in this document have been defined and if so to process them.

The method contextInitialized contains the key section of code which results in a servlet filter being created for a web application

```
IFilterConfig config = facade
            .getFilterConfig("WSCResponseFilterConfig");

config.setFilterClassName("com.ibm.wsc.wcsf.WebContainerServletFilter");
config.setDisplayName("WSC-WebContainer-IFilterConfig");
config.setName("WSC-Filter");
config.addInitParameter("traceLevel", traceLevel);
config.addInitParameter("smfUserDataKey", smfUserDataKey);
int[] dispatchMode = { 0 };
config.setDispatchMode(dispatchMode);
```

The above code creates an IFilterConfig object, and then calls various methods on this object to set various parameters, once of which is the name of the WebContainerServletFilter class.

Note the use of the addInitParameter method, this is used to pass parameters to the servlet filter.

One of the following two lines of code will be executed to set the mapping filter for the servlet filter:

```
facade.addMappingFilter("/" + contextRootMappingFilter,
config);


facade.addMappingFilter("/*", config);
```

Which one is executed depends on whether any wsc.servletFilter.contextRoot.n.mappingFilter.x type properties have been defined.

The setFilterForApp method checks to see if the current context root matches the value specified in any defined wsc.servletFilter.contextRoot.n type properties.

2.2 The SMF user data code

The code that writes the user data to the SMF record is located in the

WebContainerServletFilter class.

There are two main parts to this code.

2.2.1 Context lookup of SMF object

In the init method of this class is the following code:

```
try {
      InitialContext ic = new InitialContext();
      Object obj = ic.lookup(SmfEventInfrastructure.SEI_LOC);
      smfEventInfraObject = (SmfEventInfrastructure) obj;
} catch ( Exception ex ) {
      if ( traceLevel > 1 )
            System.out.println("WebContainerServletFilter: SMF lookup
            exception: " + ex);
      }
```

The above code performs a lookup of the SmfEventInfrastructure object. The value of this does not change while the WAS servant is running, thus this lookup is done once during init processing and stored in a class variable.

2.2.2 Write user data

The following code in the doFilter method performs the task of writing the user data to the SMF record:

```
if ( smfEventInfraObject.isSMF120Subtype9Enabled()) {
      if ( qs.length() <= 2000 )
            rc =
            smfEventInfraObject.addDataToSMF120Subtype9Record(smfUserKey
            Data, (qs).getBytes("Cp1047"));
      else {
            rc =
            smfEventInfraObject.addDataToSMF120Subtype9Record(smfUserKey
            Data, (qs.substring(0,19999)).getBytes("Cp1047"));
```

The `isSMF120Subtype9Enabled` method returns true only if the server has been configured to write SMF 120 records.

Note the use of the `getBytes("Cp1047")` this causes the query string to be written in EBCDIC to the SMF record. If you wanted the query string written in ASCII then remove this.

This link provides some more details on these methods:

http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.javadoc.doc/web/apidocs/com/ibm/websphere/smf/SmfEventInfrastructure.html

This link provides details about how to enable a server to write SMF 120 subtype 9 records:

http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.zseries.doc/info/zseries/ae/ttrb_SMFenablesmf.html

2.2.3 SMF return code meanings

We created a class called getSmfRcDesc which returns a string providing a readable error message based on the return code value from the addDataToSmf120Subtype9Record method call.

2.3 Modifying the code

The main aim of the supplied sample is to show how to use the publically available WAS API's to assign servlet filters to deployed web applications.

Using the supplied code you are free to change it to suit your particular circumstances. For example you may not want the servlet filter to write the query string to the SMF record but to perform some other function. You would then simply remove the code currently present that does this in the doFilter method and replace it with code of your own.

You may not want to use custom properties on the JVM of the WAS server to pass in settings, you can replace this with whatever method suits your environment.

--- End of Document ---