WebSphere Application Server

WebSphere Liberty: Understanding the SMF 120 Subtype 11 Record

This document can be found on the web at:
`www.ibm.com/support/techdocs`
Search for document number **WP102655** under the category of "White Papers"

Version Date: September 12, 2016
See "Document change history" on page 12 for a description of the changes in this version of the document

IBM Software Group
Application and Integration Middleware Software

Written by:
David Follis
IBM Poughkeepsie
845-435-5462
`follis@us.ibm.com`
Don Bagwell
IBM Advanced Technical Sales
301-240-3016
`dbagwell@us.ibm.com`

Many thanks go to the Tim Spewak for writing all the code and Jim Mulvey for getting in into the product.

Table of Contents

Introduction

WebSphere Liberty is....well, you probably already know that or you wouldn't be reading this. And if you don't know about Liberty, the SMF records are hardly the place to start. So we'll just skip over the background and get right into SMF.

A ~~Brief~~ History of SMF Records in WebSphere Liberty

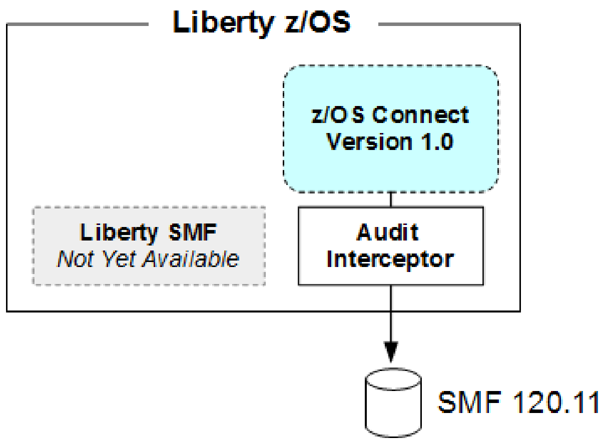
For quite a while after the initial release of Liberty the server didn't cut any SMF records itself at all. If you needed visibility to activity inside the server via SMF you needed to enable the z/OS WLM feature and assign transaction class names to incoming HTTP requests. That would cause a WLM enclave to be associated with the dispatch of the HTTP request. Ultimately that would be reflected in the SMF 72 records written by RMF and often made readable using the RMF Workload Activity Report (or similar vendor product).

That's enough to get you things like request counts and average response time and CPU time. But it doesn't tell you a lot of specifics about what actually ran. Even if you got a little carried away putting different URIs into different report classes, it is still more of a summary of what happened. That's often enough, but sometimes, especially when things are going badly, you'd like some more detail.

The first hint of SMF support in Liberty came as part of the z/OS Connect Liberty feature. When it first appeared, z/OS Connect was just another feature available as part of Liberty (and again, if you aren't familiar with z/OS Connect, this isn't the place to explain it..there's lots of material out there to explain how it fits into API Management and so forth[1]).

One of the core capabilities of z/OS Connect is the option to have interceptors get control around the dispatch of a z/OS Connect REST request. The interceptor API is documented and so you can write your own, but IBM thought it would be a good idea to provide a few basic ones to get things started. One of the interceptors that shipped with the z/OS Connect feature was called the Audit Interceptor.

The z/OS Connect Audit interceptor got control before and after dispatch and grabbed some useful information about the request and wrote an SMF record for it. It wrote a Type 120, subtype 11 record. The Type 120 record is the WebSphere SMF record and WebSphere traditional used subtypes up through 10, so that seemed the right thing to do.



The 120-11 record written by the Audit Interceptor had just two sections. A server identification section and a user-data section. The server identification section obviously had stuff in it to help you identify which server wrote the record. The information about the REST request went in the

1 For example, go to `ibm.com/support/techdocs` and search for WP102439 (Version 1) or WP102604 (Version 2).

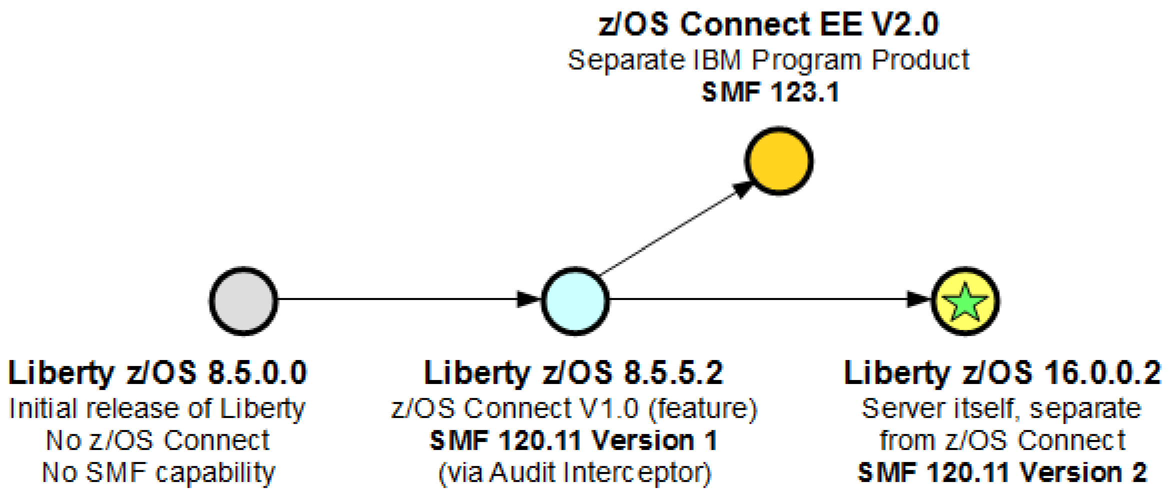
user-data section. But why would it be user data? The WebSphere traditional 120-9 record has a user data section that applications can use for their own information. But the Audit Interceptor is part of z/OS Connect which is part of Liberty..why would it be user data?

Well, user data comes with an integer tag to identify what user data it is (to help formatters figure out what to do). And the user data in the 120-9 record reserved a range of tag values for IBMs own use. To an application server, any application, even one written by IBM, is still an application and could have user data. The thought was that IBM-written applications might want to put their own information into the WebSphere SMF records and this gave us a way to do it without colliding with any customer/vendor usage of the user data API.

But still...z/OS Connect is part of Liberty, so it isn't an application, is it? Well, it was packaged as part of Liberty so kind of no, but yet it was really a servlet processing REST requests so also kind of yes. The hope was that eventually Liberty would write a real SMF record for HTTP requests with all sorts of good things in it and the Audit Interceptor could be updated to just call the user data API like any other application and add its specific stuff into the record. That's why the Audit Interceptor didn't include a lot of things in the record that you'd probably really want for a general SMF record for an HTTP request. More was coming...we thought.

As it turned out, the z/OS Connect feature in Liberty evolved into a product called z/OS Connect Enterprise Edition Version 2.0. And that product has all sorts of cool things in it way beyond what the z/OS Connect feature in Liberty has. And along the way the Audit Interceptor got rewritten to write its own SMF record that wasn't tied to the WebSphere record type (since z/OS Connect EE was its own product and not part of Websphere). The z/OS Connect EE V2.0 SMF record is the Type 123 and we're not going to talk about that anymore here.

But we still needed an SMF record for requests processed by Liberty that weren't related to z/OS Connect. And in Liberty 16.0.0.2 we finally got it done. We made a lot of enhancements to the record and, because of that, we changed the version number. The SMF 120-11 Version 1 is the record written by the original z/OS Connect V1.0 Audit Interceptor. The SMF 120-11 **Version 2** is the new record written by Liberty for any HTTP request. In summary:



And that's what we're going to talk about here.

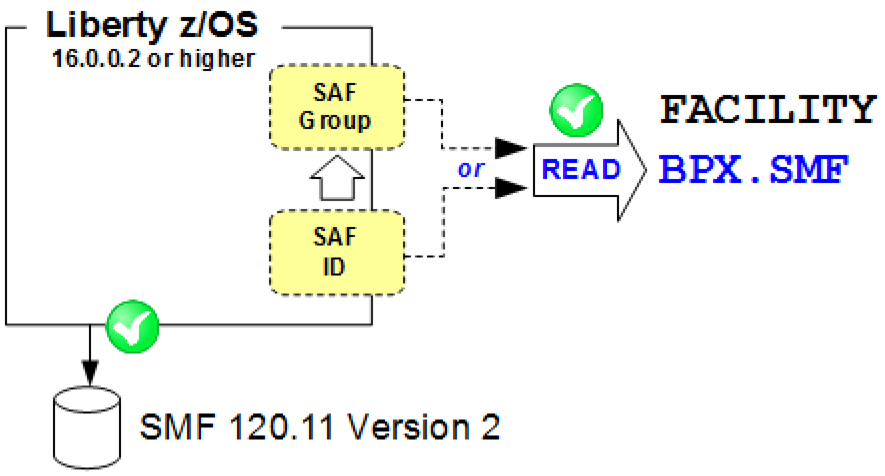
Enabling Liberty SMF Recording

In WebSphere traditional to enable SMF recording you needed to set some configuration options for the server and make sure the `SMFPRMxx` parmlib member you were using allowed SMF 120 records to be recorded (and that you didn't have an SMF exit suppressing them...).

WebSphere Liberty is the same (SMFPRMxx allows SMF 120, and no SMF exit suppressing them, and is also a bit different. Like pretty much everything in Liberty, SMF recording is an optional feature. If you add the `zosRequestLogging-1.0` feature to the featureManager clause of your `server.xml` the server will *try* to write an SMF record for every HTTP request.

But that's not all you need to do. Writing SMF records requires the calling program to be APF authorized. However, a Liberty server is generally not running as an APF authorized program. We could have made use of the Angel to get authorized (as Liberty does for other system functions). But we decided we didn't want to require the Angel to get SMF records. WebSphere traditional didn't have this concern since SMF records were written by the controller which always ran APF authorized.

Fortunately for Liberty, LE/USS provides an `smf_record( )` API that can be called by unauthorized programs to write SMF records. To make this work the identity of the server has to be permitted READ access to the `BPX.SMF` entity in the `FACILITY` class:



Therefore you also need to ensure that whatever userid (or group) the server is running under is granted this access. If the server is a started task then the started task identity is the one you want. If the server is started from the shell using the 'server' command then it will be the userid of the shell.

Formatting Liberty SMF Records

You have a Liberty server at least at level 16.0.0.2 with the `zosRequestLogging-1.0` feature installed. You configured the feature. You granted the right SAF permissions. You ran some HTTP requests through the server. Parmlib is set up to allow SMF 120-11 records and no obscure SMF exits you no longer have the source for are blocking the records from being written.

You run the SMF dump utility and behold... SMF 120 records. Now what?

We've updated our little Java program (`bbomsmfv.jar`) that formats the WebSphere SMF records to process the new 120-11 version 2 records. That's not a bad place to start just to verify you are actually getting records and the contents look right.

If you want, you can also write Java plugins for the IBM formatter to produce your own reports. Or if you use a vendor product to handle other SMF records, check with your vendor to see if they support the 120-11 records. Remember to ask about version 2 since only z/OS Connect wrote the version 1 record.

Contents – The Server Information Section

The first section in the record is always present and there is just one instance of it. This section identifies the Liberty server that wrote the record and provides some other generic information that might be useful in processing the rest of the record.

Field	*Description*	*Length*[2]
`SM120BAL`	Version	4
`SM120BAM`	System Name (CVTSNAME)	8
`SM120BAN`	Sysplex Name (ECVTSPLX)	8
`SM120BAO`	Server Job id (JSABJBID)	8
`SM120BAP`	Server Job name (JSABJBNM)	8
`SM120BAQ`	Server Stoken (ASSBSTKN)	8
`SM120BAW`	Server ASID (ASCBASID)	4
`SM120BAX`	Server Config Directory	128
`SM120BAY`	Server Product Version	16
`SM120BAZ`	Server PID	4

We start with the system and sysplex name where the server is running. The uniqueness of these names depends, of course, on how unique the names are in your environment. Next we include the JES jobname and job id of the server (i.e. `BBGZSRV` and `STC00824`). The job id is kind of unique, but the jobname might be a key to identifying the server. Maybe.

For the geeky identifiers we also include the ASID and STOKEN of the server. ASID values get reused of course, so it isn't a great identifier. But it might help correlate with other records that contain an ASID. The STOKEN is unique within the IPL which is hopefully not something that happens very often. It is less commonly used, but more unique. We also include the Unix System Services (USS) process id (PID) to help correlate with any USS related SMF data you might have.

Finally, we include the server code level (i.e. 16.0.0.2) and server configuration directory. You will notice we have **not** included the server name. The server name is really just the name of the actual directory where the `server.xml` lives. If my `server.xml` is located at:

```
/u/follis/servers/server1/server.xml
```

then the 'name' of my server is 'server1'. But if my neighbor Tim has a server at:

```
/u/tim/servers/server1/server.xml
```

then the name of his server is *also* 'server1'.

To have the real name of the server you need *both* the directory name AND the path that got you there. That's why we include the whole path to the directory where `server.xml` lives (field `SM120BAX`, "Server Config Directory"). Note that this field is a maximum of 128 characters. If the path is longer than that, we truncate on the left, keeping the likely more interesting bits towards the end of the path.

2 Length is expressed in decimal format.

Contents – The Request Information Section

This section contains information about the actual dispatch of the request. You will get one of these per record. We already know which server it ran in, so now we'll tell you about the thread it ran on. We give you both the TCB address and the TTOKEN for the thread. TCBs get re-used for different threads, but TTOKENs are unique. We also give you the USS thread ID in case you need that to correlate with something else. And just to be complete we also tell you the Java thread id.

Field	*Description*	*Length*[3]
SM120BBP	Version	4
SM120BBQ	TCB Address (PSATOLD)	4
SM120BBR	Ttoken (STCBTTKN)	16
SM120BBS	USS Thread ID	8
SM120BBT	System GMT offset (CVTLDTO)	8
SM120BBU	Java Thread ID	8
SM120BBV	Request ID	23
	reserved	1
SM120BBW	Request Start Timestamp	8
SM120BBX	Request End Timestamp	8
SM120BBY	WLM transaction class	8
SM120BBZ	Timeused CPU at start	16
SM120BCA	Timeused CPU at end	16
SM120BCB	WLM enclave delete CPU	8
SM120BCC	WLM enclave delete CPU service	8
SM120BCD	WLM enclave delete zAAP CPU	8
SM120BCE	WLM enclave delete zAAP service	8
SM120BCF	WLM enclave delete zIIP CPU	8
SM120BCG	WLM enclave delete zIIP service	8
SM120BCH	WLM enclave delete zAAP norm.	4
SM120BCI	WLM enclave delete resp. ratio	4
SM120BCJ	WLM enclave token	8
SM120BCK	User Name	64
SM120BCL	Mapped User Name	8
SM120BCM	Length of URI	4
SM120BCN	URI	128

Having the thread the request was dispatched on can be useful if you're trying to monitor how the thread pool is being used in the server. Remember though that Liberty has one thread pool that is used for everything, but you only get SMF records for HTTP requests being dispatched. Lots of other things can be happening on those threads.

We're going to have some timestamps and you might want to convert those to local time to correlate with other events. To help with that we give you the system's GMT offset taken from the CVT. And then we have the two timestamps you care about: request start and request end.

3 Length is expressed in decimal format.

Obviously you can just subtract to get response time, but you might also want to count requests that arrive in the same minute or even second to track incoming workload.

I skipped over the request id. This is a 23 byte identifier generated by the WebSphere Liberty RAS code to identify the request. I believe it is supposed to be pretty unique.

If you have the z/OS WLM feature enabled and have configured classification rules in your `server.xml` file to assign a transaction class string to the HTTP request[4], then we'll tell you what the transaction class was. The server will also wrap a WLM enclave around the dispatch of the request and, when complete, the enclave gets deleted and we'll report CPU time in the SMF record. WLM gives us back a few different CPU time values and we report everything that gets returned. This includes total CPU used, CPU used on zIIP or zAAP engines, and CPU used reported in service units. For zAAP CPU time the normalization factor is provided. The zIIP CPU time is already normalized.

WLM also tells us the response time ratio. This is a pretty cool value that tells you how this particular request did compared to the goal for the service class for which the work was classified. A value of '100' means the request exactly hit the response time goal. Bigger values means we missed the goal and small values means we beat it. The value tops out at 1000.

Finally, we tell you the actual enclave token WLM gave Liberty to keep track of the enclave. This isn't really of much use to you unless you're matching it against tracing or some other documentation. They are pretty unique (within the IPL).

If you aren't classifying requests with WLM you can still get CPU time used by the request. Liberty calls the z/OS TIMEUSED macro before and after the dispatch of the request and reports both values in the SMF record. You can subtract to determine the actual CPU time used. The values are in TOD format so (according to the documentation) bit 51 is equivalent to one microsecond.

The request section also gives you the userid associated with the request. We provide two different values. One is the received identity which is up to 64 bytes long and the other is the SAF user name we mapped that identity to, if configured to do that. The mapped identity is only eight bytes long.

Finally we include up to 128 bytes of the URI for the request. The length field before the URI tells you how much of those 128 bytes are actually data, in case the URI is shorter.

Contents – The Classification Data Section

This section contains the data that was used in processing the HTTP request classification rules, if any, in your `server.xml`. This is how we determined the transaction class name reported in the request section.

Field	*Description*	*Length*[5]
`SM120BDA`	Version	4
`SM120BDB`	Type	4
`SM120BDC`	Length of data	4
`SM120BDD`	Classification data	128

This section *repeats*, once for each piece of data we use to classify the request. There will be a section for the URI, which is probably the same as the URI we reported in the request section. There will be an instance of the section for the target hostname. This is probably the hostname for

4 Go to `ibm.com/support/techdocs` and search for WP102110. That Techdoc illustrates how to do WLM classification.

5 Length is expressed in decimal format.

the Liberty server, but depending on routing it might not be. And finally an instance of the section reporting the target port number.

Contents – The Network Data Section

The network data section contains information about the network interactions the server had for this request. We provide the origin of the request and the port number used to make the request. If the requests all come through a gateway, this might be the same for every request.

Field	*Description*	*Length*[6]
`SM120BCR`	Version	4
	Reserved	8
`SM120BDI`	Response Bytes	8
`SM120BCS`	Target Port	4
`SM120BCT`	Remote Port	4
`SM120BCU`	Length of remote address	4
`SM120BCV`	Remote Address	40

We also give you the target port number. This is probably the same as the target port in the classification section. Remember that a Liberty server can listen on more than one port. Different users might be told to use different ports to reach the same server and knowing the target port that was used might help distinguish SMF records for one group of users from another.

Finally, we report the number of bytes received for the request and the number of bytes sent for the response. Some customers may want to do chargeback for this. More often it is just used as a diagnostic. Requests that use excessive amounts of CPU might have been handling an unusually large request or generated an unusually large response. Giant requests or responses can also have implications to network performance and having information about the general size (and the time of day when they occur) may help you plan network capacity.

Contents – The User Data Section

The user data section is only present if the application (or a servlet filter) called the SMF user data API on the dispatch thread during execution of the request. A call can supply up to 2K of data and an integer 'tag'. The tag is intended to identify what user data it is (in case different applications use the API) so your post-processor can correctly parse the data.

Field	*Description*	*Length*[7]
`SM120BAR`	Version	4
`SM120BAS`	User Data Tag	4
`SM120BAT`	User Data Length	4
`SM120BDH`	User Data	2048

The user data can, of course, be anything you want that is interesting to the application or to whatever you are doing in post processing. Most commonly it is additional information about the nature of the request that can help determine what the request was for. For example, a single servlet might have several functions and which one is driven depends on a parameter value provided by the caller. The user data is a handy place to put the value of that parameter.

6 Length is expressed in decimal format.
7 Length is expressed in decimal format.

Depending on how SMF data is handled at your shop, be careful not to put sensitive information (like account numbers) in the user data unless you are sure access to SMF data is restricted appropriately.

Adding user data can be done like this:

```
InitialContext ic = new InitialContext();
try {
    UserData userData =(UserData)
               ic.lookup("com/ibm/websphere/zos/request/logging/UserData");
    int rc = userData.add(65535,"my request specific data");
} catch (NamingException e) {
}
```

If two calls are made to the `add()` API with the same tag value, the new data replaces the old. Up to five different user data blocks may be added to one request.

Document change history

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

September 7, 2016	Initial Version
September 12, 2016	Add document number

End of WP102655