WebSphere Application Server for z/OS V7 and V8

Understanding SMF Record Type 120, Subtypes 9 and 10

This document can be found on the Web at:

www.ibm.com/support/techdocs

Search for document number **WP101342** under the category of "White Papers"

Version Date: January 12, 2022

See Document change history on page 33 for a description of the changes in this version of the document

IBM Software Group

Application and Integration Middleware Software

David Follis
IBM Poughkeepsie
845-435-5462
follis@us.ibm.com

Edited by Wayne O'Brien
IBM Poughkeepsie

Many thanks go to ...Tim Spewak, Steve Dittmar, Mike Ginnick, Robert Alderman, John Hutchinson, Don Bagwell, and loads of others

Introduction **4**
Structure of the subtype 9 record **4**
Why have platform neutral sections? 5
A word about field names 5
And another about versioning 5
Header and triplets **6**
Platform neutral server information **6**
z/OS server information **6**
MVS identity 6
WebSphere identity 7
Time offsets 7
Service level 8
Platform neutral request information **8**
z/OS request information **9**
Times 9
Servant identification 10
CPU usage 10
Enclave CPU times 10
Global transaction identifier 11
Timeouts and transaction classes 12
Flags 12
z/OS formatted timestamps **13**
Network data **13**
Classification data **13**
Security data **14**
CPU usage breakdown **15**
User data **17**
Operational considerations **17**
SMF browser updates **18**
Interval records **20**
Conclusion **21**
Updates for Version 8! **22**
Things That Moved Around or Changed **22**
Affinity Routing Information **22**
Granular RAS Control Information **24**
SMF Type 120 Subtype 9 Async Work Records **25**
New SMF Type 120 Subtype 10 – Outbound Requests – WOLA (8.0.0.1) **27**
Outbound Request Information 28
WOLA Outbound Request Information 28
Outbound Request Transaction Information 29
Outbound Request Security Information 30
WOLA Outbound to CICS Information 30
WOLA Outbound OTMA Information 30
Operations 30
Document change history **32**

Introduction

In z/OS, the system management facilities (SMF) component allows you to gather and record data for evaluating system usage. WebSphere Application Server has logged activity data through SMF records since Version 4 of the product.

SMF reserves record type 120 for WebSphere activity data. Record 120 includes a number of subtypes, each of which contains a subset of the data. This fragmented view of the data is due to internal divisions in the product. Some record 120 subtypes are created by the server runtime (subtypes 1 and 3), while others are created by the Web containers and EJB containers (subtypes 5, 6, 7, 8). Because each subtype provides only a partial view of the activity, you need to correlate several subtypes to get a more complete picture of what the server is doing.

In WebSphere Application Server for z/OS Version 7, we introduce subtype 9, a new subtype that presents a unified picture of the server activity. The subtype 9 record collects most of the data currently spread across the other subtypes, plus new information, such as how much zAAP CPU the server uses in processing a request. WebSphere creates one subtype 9 record for every request that the server processes — for both external requests (application requests) and internal requests, such as when the controller "talks to" the servant regions.

In designing the subtype 9 record, we consulted many z/OS customers. Most of you told us that you would prefer to have all of the server activity data collected in one record. Thus, was born the subtype 9 record, which is intended to provide you with all of the server activity data you need in one place. The other record 120 subtypes still work as before, but we hope you will not need them any longer.

In this IBM White Paper, we take a closer look at the new subtype 9 record. We examine its structure and content, its related operational issues, and explain how you can make the best use of it.

UPDATED!

This paper has been updated to include new sections at the end describing the changes and enhancements made to WAS SMF processing in Version 8! This includes updates to the 120-9 record and the introduction of the 120-10 record for outbound WOLA requests.

Structure of the subtype 9 record

The subtype 9 record contains a variable number of sections as shown in Table 1. Structure of SMF record type 120, subtype 9).

Subtype 9 section	Number of occurrences	Configurable?
Header	1	No
Triplets	1	No
Platform neutral server information	1	No
z/OS server information	1	No
Platform neutral request information	1	No
z/OS request information	1	No
z/OS formatted timestamps	0 – 1	Yes
Network data	0 – 1	No

© 2008,2011, IBM Software Group, Application and Integration Middleware Software, Poughkeepsie, NY

Classification data	0 – 5	No
Security data	0 – 3	Yes
CPU usage breakdown	0 – 30	Yes
User data	0 – 5	No

Table 1. Structure of SMF record type 120, subtype 9

In the subtype 9 record, some sections appear only once, while others can appear multiple times or not at all. A section might not appear if its information is not applicable to the type of request being reported, or because the server is not configured to write the section. In this document, we note how many instances of each section can appear and, for an optional section, what controls its appearance.

Why have platform neutral sections?

In Table 1, notice that the server and request information consists of a platform-neutral section and a z/OS-specific section. You might wonder why we included platform-neutral sections in subtype 9 when SMF records are available only on z/OS? Are we planning to provide an SMF-like capability in WebSphere on other platforms?

Not in the near future. If we ever did, however, it would be useful if the record structure included both platform-neutral sections and platform-specific sections. By designing the subtype 9 record this way, we were just laying some groundwork, in case a cross-platform WebSphere chargeback mechanism evolves someday.

A word about field names

Field names in SMF record type 120 use the naming convention of a prefix (SM), followed by the record number (120), then three unique characters. In the existing record 120 subtypes, the three characters are somewhat meaningful. For example, the *CSH* in field name **SM120CSH** stands for communications session handle.

In designing the new subtype 9 record, we considered choosing meaningful character identifiers, but also needed to avoid collisions with any identifiers we used in the previous record 120 subtypes. To ensure unique field names, we included the subtype number— "SM1209"— as a prefix for subtype 9 field names.

This approach left us with just two characters to identify the field. Finding a meaningful three-character abbreviation for a field is hard enough, but with just two characters? No way.

In the end, we just labeled the fields with ‘AA’, ‘AB’, and so on, alphabetically. We avoided naming conflicts, but the names are meaningless.

And another about versioning

Each section in the subtype 9 record contains a version number and reserved space. The reserved space allows us to add information to a section without making it bigger. If we change the contents of a section, even if we just use reserved space to include additional information in the section, we increment the version number for the section.

Given that the size of the section is located in the triplet at the top of the record and the version number tells you how to parse the section, you might wonder why we need the reserved space. Technically, we don’t. However, some programs that analyze SMF records do not look at the length field in the triplet, and instead use the fixed offsets to find sections in the records.

It is not a good practice, but it happens and, when it does, a change in the size of a section breaks the program. Therefore, we try not to change the size of a section very often and we manage that by including some reserved space.

© 2008,2011, IBM Software Group, Application and Integration Middleware Software, Poughkeepsie, NY

That said, always use the triplets at the top of the record to locate the sections and use the version number in the section to determine which fields are present and useable.

Header and triplets

All SMF records begin with a standard header that includes the record type and subtype, a timestamp to indicate when the record was written, and some other details. Following the header is a set of "triplets" (offset, length, number) that tell you what parts of the record are present and where to find them.

Platform neutral server information

The fields in this section help to identify which WebSphere server processed the request. As mentioned earlier, the platform neutral section could apply to a WebSphere server running on any platform. Thus, the fields are intended to identify the server.

Most server names in WebSphere have a ‘short’ and a ‘long’ form. The long form is usually more descriptive, but in some cases, the name is used as an identifier in an API that has a length restriction; therefore, short and long versions of the names have evolved.

This section begins with the cell short name (field **SM1209BH**). This is followed by the server node, cluster, and short name (fields **SM1209BI**, **SM1209BJ**, and **SM1209BK**, respectively). These four names should, in combination, uniquely identify the server in your environment.

Ah, but what instance of that server? It could matter to you that the server restarted overnight and records written for requests processed before the restart versus after the restart might be interesting. Therefore, field **SM1209BL** contains the z/OS UNIX Systems Services process identifier for the controller process. Most platforms have a process identifier of some sort, and that is why the field **SM1209BL** resides in the platform-neutral section.

Lastly, you might it useful to know the release and maintenance level of the server that wrote the record. This information is provided in four one-byte fields, which (in combination) provide the version and maintenance level:

- **SM1209BM**
- **SM1209BN**
- **SM1209BO**
- **SM1209BP**

For example, a Version 7 server at maintenance level 5 is called Version 7.0.0.5. Here, these four fields would contain a seven, a zero, a zero, and a five, respectively.

We hope that breaking up the information this way makes it easier for you to sort or compare records at different release or maintenance levels.

z/OS server information

The fields in this section help to identify the server that wrote the record with respect to the z/OS operating system, or use server identifiers that only apply to WebSphere Application Server on z/OS.

MVS identity

This section begins with two fields that identify where in your enterprise the server is located:

- **SM1209BR** contains the system name. This value is obtained from the CVTSNAME field of the MVS CVT control block. The CVT field is padded on the right with blanks and is not null terminated.

© 2008,2011, IBM Software Group, Application and Integration Middleware Software, Poughkeepsie, NY

- **SM1209BS** contains the sysplex name. This value is obtained from the ECVTSPLX field of the MVS ECVT control block. This field is padded on the right with blanks and not null terminated.

The next four fields help to identify the controller region from an MVS perspective:

- **SM1209BT** is the job name, which is obtained from the JSABJBNM field of the JSAB control block.
- **SM1209BU** is the job ID, which is obtained from JSABPREF. The job ID can be used to find the job output in the JES spool.
- **SM1209BV** is the stoken, which is obtained from the ASSBSTKN field of the ASSB control block (one per address space). An stoken uniquely identifies an address space for the life of an IPL. In other words, it will not be reused unless you IPL (reboot) the operating system.
- **SM1209BW** is the address space identifier (ASID) of the controller's address space. This value is obtained from the ASCBASID field of the address space control block (ASCB). An ASID can be reused, but it is a shorter (2 bytes) and more manageable identifier for the process.

WebSphere identity

The next three fields help identify the server from a WebSphere perspective.

- Fields **SM1209BX** and **SM1209BY** are the universally unique identifiers (UUIDs) for the server. The first UUID (**SM1209BX**) is the cluster id. All servers in the same cluster share the same cluster id. The second UUID (**SM1209BY**) is the server id. No other server in the universe should have the same server UUID. No other cluster should have the same cluster id. Therefore, you can use these fields to determine the server from which the record came.

  As a 20-byte binary value, a UUID might not be the most convenient thing to use, but it is probably better than having to compare all four of the server names (cell, node, cluster, and server) which might still not be unique. Why aren't the UUIDs in the platform-neutral section if they are WebSphere identifiers? Because a WebSphere Application Server on z/OS UUID is 20 bytes long and a WebSphere Application Server on distributed UUID is only 16 bytes long and we did not want to confuse things. Too late, huh? :)
- Field **SM1209BZ** is the daemon group name (eight bytes long, padded on the right with blanks and not null terminated). In most cases, this is the same as the cell short name. WebSphere on z/OS has an extra process, the daemon, which does not exist on distributed systems. The daemons communicate with each other when they are in the same cell. However, we did not want to be tied to the cell name when knowing which daemon to talk to, therefore we made up another name, the daemon group name, to identify which daemons should be friends. Mostly it is the same as the cell name, but because there are a few cases where it is different, we included it in the SMF record, in case it matters.

Time offsets

The next fields in this section do not really have anything to do with identifying the server, but they are platform-specific so this seemed like a good place for them. This set of four fields indicates the time zone offset from Greenwich Mean Time (GMT) as used by this server. There are two places you can get that value from; we get and report both values in case they are different. You might use these values to adjust time stamps found later in the records (which are in GMT) to an appropriate local time. That might be important if you do chargeback differently, based on the time-of-day when resources were used. Users would expect to be charged based on their local time, not based on the time in England (unless, of course, the users ARE in England).

Anyway, the fields **SM1209CA**, **SM1209CB**, and **SM1209CC** contain the hours, minutes, and seconds of the offset from GMT, as reported by the z/OS Language Environment callable service, CEEGMTO. Be careful with these values. While the hours and minutes can be combined to form the offset, the 'seconds' value is the entire offset in seconds. For example, if the offset is one hour exactly, the hour's value with be one, the minute's value is zero, and the second's value is 3600.

Also, the CEEGMTO service is available only for callers in 31-bit mode. If the server is running in 64-bit mode, these values are not available. The fields are set to zeroes and a flag (**SM1209FJ**) is turned on to indicate that the zeros in this field are not valid GMT offsets.

For more information about the CEEGMTO service, see the book *z/OS Language Environment Programming Reference*, SA22-7562.

For working with TOD-format timestamps, you might find the value in **SM1209CD** to be more useful. This field contains the GMT offset taken from the CVT field CVTLDTO. To get the local time, add this value to a TOD-format timestamp.

Service level

When the server starts up, it issues message BBOM0007I to indicate the service level. The message includes a numeric identifier and the build identifier in parenthesis. The platform neutral section of the subtype 9 record contains the numeric identifier for the level (for example *7.0.0.5*). The build identifier is included in this section, in field **SM1209CE**.

There is generally a one-to-one correspondence between the numeric identifier and the build level, but it seemed like a good idea to include both in the subtype 9 record, just to be complete. You might notice that by squeezing the service level indicator into an 8-byte field, we left off the last digit of the level. A level of *ww0743.40*, for example, appears in the record as just *ww0743.4*.

Platform neutral request information

The fields in this section describe the request. We begin, as before, with platform independent information and then move on to the z/OS specific stuff in the next section.

First, we would like just a bit more information about where the request ran. On z/OS, a server is divided into several processes: A controller and one or more servants. Previously, we mentioned that field **SM1209BL** contains the controller process id. Here, field **SM1209CG** provides the process id for the servant in which the request ran. Because the servant is a "z/OS thing," this field could have gone into the next section, but process ids exist on all platforms, so we placed it here.

We would also like to know which the thread on which the request ran inside the servant; therefore, we report the task id (as returned by the pthread_self API) in field **SM1209CH**. While you probably aren't interested in which thread got used to process a request, you might want to know how many threads are being used, and you could use this value (or the TCB address in **SM1209CV** in the next section) to find out how many of the dispatch threads configured for a servant are actually being used.

Now let's look at some resource usage reporting. Field **SM1209CI** contains the CPU time used by the dispatch thread while the request was in dispatch. We use the TIMEUSED macro to get the CPU time on the TCB before and after dispatch and then subtract to yield CPU time used by the thread in dispatching the request. The value is shifted right to put the value reported here in microseconds. We placed this field in the platform neutral section because most platforms can report CPU time used. The z/OS specific CPU times, such as zAAP time, reside in the z/OS specific section.

Suppose, as a general metric, you need to know how many requests are completing successfully. Or perhaps, your method of chargeback considers whether the request worked. If so, you will be interested in **SM1209CJ**, an 8-byte field that contains the exception minor code for the request. If the field is zero,

no exception was returned from dispatch of the request (which doesn't mean it "worked," it just did not throw an uncaught exception).

Be careful about building analysis tools that rely on specific values showing up in this field. Minor codes are not fixed and the same error could produce a different minor code due to a code change delivered in a WebSphere maintenance level.

In addition, if a minor code is present, it might mean that the request never dispatched or timed out and was terminated. Here, some of the data normally in the SMF record is not present. That's because some SMF data is collected in the servant region and copied into the controller region when dispatch completes. If the dispatch never completes (or never even starts), the data is not collected and will not be collected in the record.

The last field in the platform neutral section is **SM1209CK**, which indicates the request type. This field is just a number that indicates how the request got to the server. If the request arrived over an HTTP connection, for example, the "type" is 2. Other values represent other paths to the server.

Some work requests are generated internally, which happens when the controller region "talks to" a servant. Internal requests are the result of various administrative or other internal operations. You could choose to recognize internal requests and charge them to "overhead" for the server, or write an SMF exit routine to suppress writing records for internal requests.

z/OS request information

This section contains more fields than any other section. It includes the really interesting stuff.

Times

We start with timestamps. These can be useful in determining when a request ran, and also in determining how well the server is responding to different workloads throughout the day. As a request moves through the major phases of dispatch, SMF collects the following timestamps for the subtype 9 record.

Field	Data	Description
SM1209CM	Arrival	The time when the controller first saw the request.
SM1209CN	Queued	The time when the controller placed the request on the WLM queue.
SM1209CO	Dispatched	The time when the request was selected from the queue by a thread in the servant and dispatch began.
SM1209CP	Complete	The time when the container returned from dispatching the request and the response (if any) was copied to the controller.
SM1209CQ	Finished	The time when the response (if any) was sent and the request was cleaned up in the controller. This time should be about the same as the time the SMF record was written.

These timestamps are in TOD (STCK) format. Some request types, such as an MDB, do not have a response, but the flow of control is still the same and all of the timestamps apply. If a request times out on the z/OS Workload Management (WLM) queue waiting to dispatch, the dispatch started and completed timestamps (**SM1209CO** and **SM1209CP**) are set to zero.

© 2008,2011, IBM Software Group, Application and Integration Middleware Software, Poughkeepsie, NY

Servant identification

This section of the subtype 9 record provides the following data:

- **SM1209CR**, the job name of the servant in which the request was dispatched
- **SM1209CS**, the job ID of the servant, for example, STC12345
- **SM1209CT**, the stoken of the servant
- **SM1209CU**, the ASID of the servant.

This section also contains the TCB address of the thread where the request was dispatched (**SM1209CV**) and the ttoken of the thread (**SM1209CW**). A ttoken is a unique identifier for a thread, much as an stoken is a unique identifier for a process (address space). These identifiers are only re-used after an IPL of the z/OS image. You might find these values useful in grouping records together to determine how well you are using the servants and threads you have defined in the server.

CPU usage

The TIMEUSED macro also allows us to determine the CPU time used for "standard" and “non-standard” CPs, such as zAAPs. The values are obtained before and after dispatch, and WebSphere does the subtraction and reports the results in the fields **SM1209CX and SM1209CI**.

Note that the CPU time in the platform-neutral section (**SM1209CI**) is all of the CPU time, zAAPs included, while **SM1209CX** is just the time on nonstandard (specialty) CPs. To get time spent on standard CPs, you have to subtract the latter from the former. Remember that both values are shifted, so you will be working in microseconds. This is different from the enclave CPU times we will talk about shortly. Also note that while SM1209CI reflects the actual value returned by TIMEUSED (subtracting the after-dispatch value from the before-dispatch value), SM1209CX reports a value not directly returned by TIMEUSED. The macro returns time on standard CPs only and WebSphere subtracts this value from what we report in SM1209CI to report specialty engine CPU time.

Further note that the call to get the enclave CPU time occurs in a slightly different spot in the dispatch path than the call to TIMEUSED. Therefore, the values might be slightly off from each other.

Enclave CPU times

The field **SM1209CY** reports the enclave token for the WLM enclave under which the request was dispatched. Enclaves can be propagated around, so it is possible for two different servers on the same system to process requests under the same enclave. If so, it usually means that an application in one server made a call to an EJB in another server. Enclave CPU times include time spent in both servers. The enclave token allows you to correlate the SMF records from the two servers.

The next fields come from a call to IWMEQTME made after dispatch of the request is complete. All of the returned values are reported. Some of these values may seem redundant with other values reported later in the record, or may seem useless in a WebSphere servant. In designing the subtype 9 record, we opted to report everything, rather than trying to guess what might be valuable.

Field	Data	Description
SM1209DA	CPU Time	Enclave CPU time used to this point
SM1209DB	zAAP time	Enclave CPU time on zAAP used to this point
SM1209DC	zAAP on CP	Enclave CPU time eligible to run on zAAP but which ran on regular CPs, to this point
SM1209DD	zIIP on CP	Enclave CPU time eligible to run on a zIIP but which ran on regular CPs, to this point

SM1209DE	zIIP eligible	Enclave CPU time eligible to run on a zIIP regardless where it ran, to this point
SM1209DF	zIIP time	Enclave CPU time on zIIP to this point
SM1209DG	zAAP normalization	zAAP normalization factor (see the documentation for IWMEQTME)

Each of the time descriptions says "to this point." That's because the enclave can continue to be used for other requests in some circumstances and these values are just "to this point." For example, if a request comes into WebSphere under a transaction, an enclave is created to dispatch the request. However, the enclave won't be deleted when the request completes because the transaction is still around. Another method could come in under the same transaction, in which case it would be dispatched under the existing enclave for that transaction.

Only when the transaction completes, is the enclave deleted. When the subtype 9 record is written for each request, the enclave CPU times are reported "to this point." The record written for the final request would thus have larger values. You can tell the enclave is being re-used because the enclave token (**SM1209CY**) is the same. You can use the transaction identifier (more about this later) to recognize requests that are part of the same transaction.

If the enclave is deleted at the end of processing for the request, the IWM4EDEL service returns information about CPU used by the work dispatched under the enclave. Again, we report all of the values returned by this API, even if it appears that some of them might never apply to a request dispatched inside the server.

Field	Data	Description
SM1209DH	CPU time	Enclave CPU time
SM1209DI	zAAP time	Enclave CPU time on zAAP
SM1209DJ	zAAP normalization	zAAP normalization factor
SM1209DK	zIIP time	Enclave CPU time on zIIP (normalized)
SM1209DL	zIIP service units	Enclave CPU service units on zIIP
SM1209DM	zAAP service units	Enclave CPU service units on zAAP
SM1209DN	CPU service units	Enclave CPU service units
SM1209DO	Ratio	Response Time Ratio

For information about what all of these values mean, see the IWM4EDEL service. Note that the zAAP parameters for both IWM4EDEL and IWMEQTME were added in service. If you do not have the correct service level applied, the fields whose values WebSphere cannot obtain are set to negative one (all 'F's). If the enclave is not deleted at the end of the request, these fields are set to zeroes.

In addition, the enclave delete service reports CPU usage in service units and in absolute time. We included this data in the record because some of you told us that you prefer to do chargeback based on service units rather than actual time.

Global transaction identifier

If the request was processed as part of a global transaction, it was assigned a global transaction identifier (GTID), included in field **SM1209DQ**. You can use the GTID to correlate records from requests that were processed as part of the same transaction.

© 2008,2011, IBM Software Group, Application and Integration Middleware Software, Poughkeepsie, NY

Timeouts and transaction classes

The field **SM1209DR** contains the timeout value used for the request in seconds. Each request that is received can be monitored to be sure it completes on time. The timeout value that is used is part of the server configuration and can vary depending on the type of request. Because you might find it useful to know what value was actually used, we included it in the subtype 9 record.

If a WLM enclave is created for the request, WLM needs to classify the request to determine what service class and report class to associate with the enclave. WebSphere uses two attributes to classify a request. The first is the cluster name (known to WLM as the collection name) and the second is a transaction class. Field **SM1209DS** contains the transaction class name that was provided to WLM as part of the classification data.

Flags

The four bytes following the transaction class are reserved for flags. Only a few bits are presently used and the rest are reserved for good ideas we haven't had yet.

Let's go through each flag and see what it tells us:

Field	Mask	Description
SM1209DU	0x80	Enclave created by this server
SM1209DV	0x40	Timeout value from external source
SM1209DW	0x20	Transaction class from external source
SM1209DX	0x10	One-way request
SM1209DY	0x08	CPU Usage section overflow
SM1209DZ	0x04	Request queued with affinity to a servant
SM1209FJ	0x02	CEEGMTO service not available in 64-bit

Flag **SM1209DU** is set if the controller region created an enclave for the request. The flag is usually set, unless an enclave token is received as part of the request. This situation typically happens for IIOP flows between servers located on the same z/OS image.

Flag **SM1209DV** indicates that the timeout value (**SM1209DR**) was received from an external source.

Flag **SM1209DW** indicates that the transaction class (**SM1209DS**) was received from an external source and was not determined by the server itself.

Some requests are called one-way requests; these do not require a response. The client sends the request and continues on, never looking for any acknowledgement that the response completed. If the request is a one-way request, flag **SM120DX** is set.

Later in this document, we discuss the subtype 9 section that contains CPU usage information broken down as the request flowed through the containers. To keep the SMF record from getting too large, the number of instances of that section is limited. If the number of sections required for all the data exceeds the limit, the extra information is discarded and flag **SM120DY** is set.

Flag **SM1209DZ** is the affinity bit. When a request enters the server, WebSphere and WLM have to determine to which servant region the request should be routed. In some cases, the request might need

© 2008,2011, IBM Software Group, Application and Integration Middleware Software, Poughkeepsie, NY

to be directed to a particular servant. For example, if a previous request from this same client established an HTTPSession in memory (say a "shopping cart"), a subsequent request might need to get back to the same servant to find the HTTPSession data. Here, WebSphere tells WLM which servant to send the request to, and this bit in the subtype 9 record is set. If you are trying to understand why some servant regions get more requests than others do, understanding how many are forced to run where they do is important.

As we mentioned earlier, flag bit **SM1209FJ** indicates that the CEEGMTO service is not available and therefore fields **SM1209CA**, **SM1209CB**, and **SM1209CC** do not contain usable data. This is probably because the server is running in 64-bit mode.

z/OS formatted timestamps

This section is an optional section. We discuss configuring these under "Operational considerations." If the timestamp section is not configured, the triplet values in the record header for this section are set to zeroes.

The data contained in this section consists of the same timestamps kept in the previous section. The fields **SM1209CM**, **SM1209CN**, **SM1209CO**, **SM1209CP**, and **SM1209CQ** are repeated here, but as formatted EBCDIC strings. The format is `YYYY/MM/DD hh:mm:ss.xxxxxx`.

If having easily readable strings in the record is useful to you, turn this section on. However, because it is possible to format the STCK format timestamps in post-processing, turning it off saves 132 bytes per record. There is also a small amount of overhead in formatting the timestamps, which is added to the cost of processing each request.

Network data

IIOP and HTTP requests arrive through a TCP/IP connection or over a local connection from another address space on the same z/OS image. For those requests, this section is present and contains information about that connection.

The fields **SM1209EG** and **SM1209EH** indicate how many bytes of information were received and sent over the connection for this request.

The **SM1209EI** field contains the server port number to which the client is connected. If the client is local and connected without TCP/IP, this field is set to negative one (all ‘F’s).

The origin of the request (usually a host and port) is contained in a string in field **SM1209EK**. The string is truncated at 128 characters if it exceeds that length. The length of the data in the field is kept in field **SM1209EJ**. For a local request, the string indicates the job name and ASID of the requestor because the request did not get to WebSphere over TCP/IP.

Classification data

As mentioned previously, WebSphere provides a transaction class to WLM to help it classify the request into a service class and a report class. WebSphere determines the transaction class by mapping attributes of the request against a tree of classification rules kept in the XML file pointed to by the environment variable **wlm_classification_file**. This section of the SMF record contains the attributes of the request that would be used in determining the transaction class name. Note that this section of the record is filled in even if you aren’t using the XML file to determine a transaction class name.

What's in there? Well it depends on the type of request being classified. For an IIOP request, the SMF record contains the application, module, and component name of the EJB, along with the class name and mangled method name. A mangled method name is a combination of the actual method name and the parameter signature merged into a string (it is the string the tie uses to figure out which EJB method to invoke). For an HTTP request, the URI and target host and port is included. For an MDB, the listener port and selector are included.

The section repeats with one attribute per section. Depending on the type of request, you might just have two sections or you might have as many as five sections. A tag (**SM1209EM)** tells you what type of attribute is contained in the section. Values 1 through 5 are used for IIOP requests. Values 6 through 8 are used for HTTP requests. Values 9 and 10 are used for MDBs received from the MQ listener in the controller.

SM1209EM value	Meaning
1	Application Name
2	Module Name
3	Component Name
4	Class Name
5	Mangled Method Name
6	URI
7	Target Host
8	Target Port
9	Message Listener Port
10	Selector

You can use the data in this section to verify that the WLM classification XML file is working as expected. That is, you can examine the transaction class (**SM1209DS**) and ensure that a particular set of attributes is yielding the right transaction class.

Regardless of whether you are using the classification XML file, this section is your chance to find out what the request really is. You can use this information to determine which application is being driven and perhaps do appropriate chargeback processing or usage profiling.

Security data

The security section provides information about the identity of the user who made the request being recorded. There can be up to three instances of this section, depending on what security information is available. Each instance contains an identity string.

To understand the different identity types that can be contained in this section, we need to consider a request chain. A user somewhere generates a request which flows to a server. The application running in that server may drive a request to another server, and so forth through a series of applications and servers. As the request flows from server to server the identity involved may change as various containers impose different rules about the identity being used. For example, it is possible that the identity received by a server might not be the same identity used to invoke an EJB the request wanted to run.

© 2008,2011, IBM Software Group, Application and Integration Middleware Software, Poughkeepsie, NY

Depending on the security configuration, the information for this section might not be available. If it is, this section reports up to three identity strings (see Figure 1. Identity propagation). The first possible identity is the server identity. This identity is usually a string formed by concatenating the cell, node, and server names, like this: "server:" + long cell name + "_" + long node name + "_" + long server name. For example, the server identity string might be “server:h2cell_h2nodeb_h2sr01b”. You can change this to be the SAF userid for the servant region started task by configuring the “User Identity for the z/OS Started Task” in Security->Global Security->User Account Repository->Available Realm Definitions->Local Operating System.

The second identity is the received identity. This is the identity that was received by the server writing this record. The third identity is the invocation identity. This is the identity that was used to actually run something on this server. The identities might all be the same or they might not.

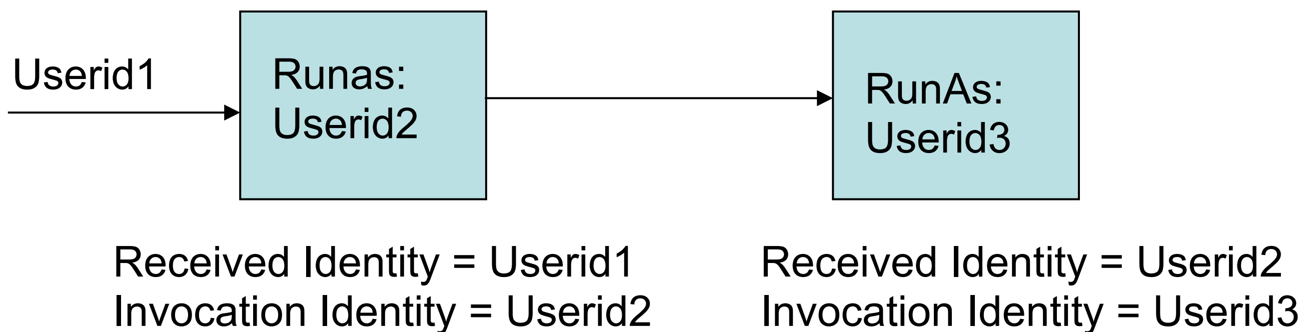

Figure 1. Identity propagation

The information in this section could be used as part of a chargeback system to bill for usage based on who was using the resources.

CPU usage breakdown

The classification section tells you what the request was, and the request information sections tell you how much CPU was used and how much elapsed time the request took to execute. However, how do we learn what the request actually *did*?

Whenever a part of your application calls another part in a way that requires the Web container or EJB container to get involved, WebSphere can use that involvement to gather information. For example, if an HTTP request drives a servlet which, in turn, calls an EJB, both the Web container and the EJB container are involved. The information reported in the other sections tells you how much CPU was used to process the whole request, but what about just the EJB?

The CPU usage breakdown section tells you that. However, gathering the information at these points and compiling it into the SMF record section requires some effort. For this reason, we made this section optional. To avoid the overhead associated with this processing, turn off this part of the record. When turned off, the triplet at the top of the record is set to zeroes.

In addition, to avoid compiling a huge mass of data, we keep only information for the first 30 unique things that require container involvement. For example, if a servlet calls 35 different EJBs, we keep information about only the first 29 (29 EJBs, plus 1 servlet). However, if the servlet calls the same EJB

repeatedly, we keep data for both the EJB and the servlet. If we discard data, the flag bit we mentioned earlier is set (**SM1209DY**).

In the existing SMF 120 records where container CPU time was reported, each entry reported time used in that object (servlet, bean, or whatever) and in anything it called. In the subtype 9 record, container CPU usage is reported only for time spent in the particular object. Time spent in other objects it calls is subtracted out and reported separately.

An example is probably in order. In Figure 2. CPU usage breakdown, suppose that an HTTP request drives Object A, which calls Object B, which calls Object C twice before unwinding back to Object A. Then, Object A calls Object B again, which in turn calls Object C again before again unwinding back to Object A. Finally, Object A calls Object C directly. In the figure, the numbers in the boxes represent time spent in the routine.

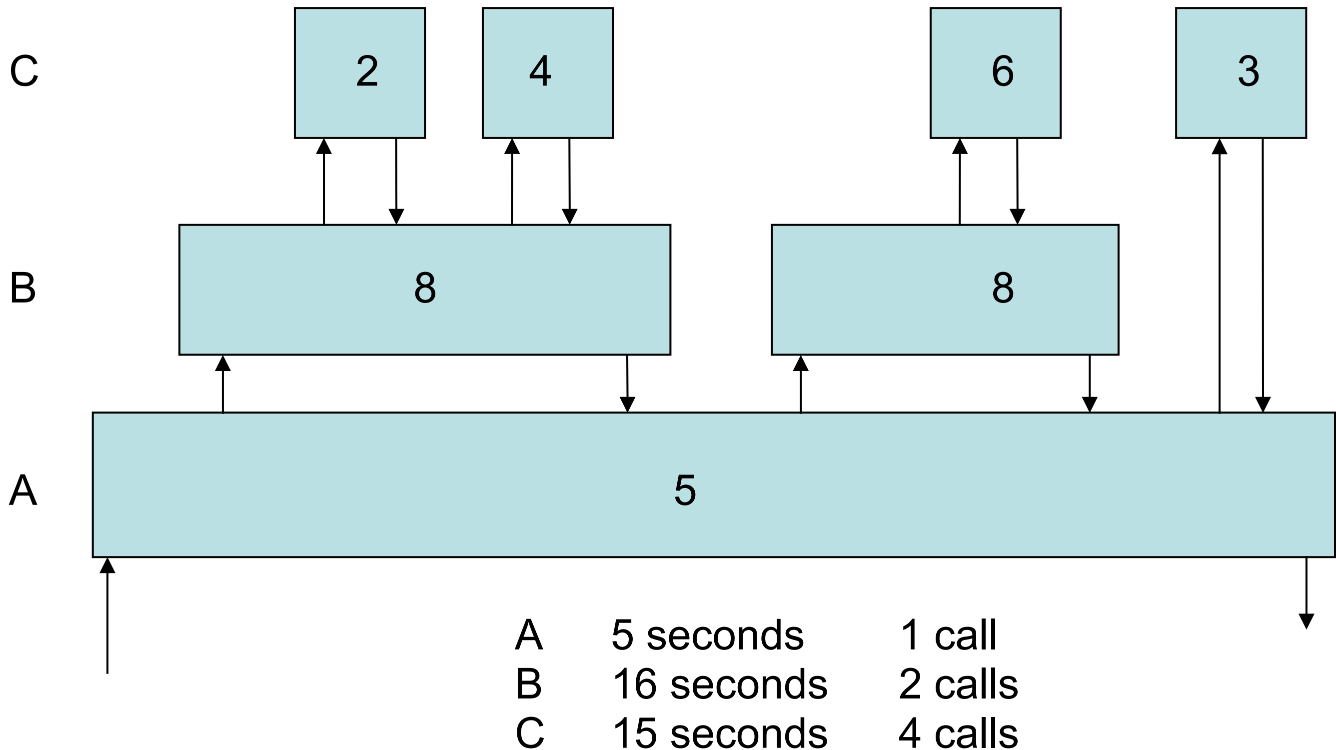

Figure 2. CPU usage breakdown

The CPU usage section contains three sections, one each for A, B, and C. The entry for ‘C’ contains the total of CPU time and elapsed time spent in Object C, as well as a count of the number of times it was called (**SM1209EW**).

The entry for ‘B’ indicates it was called twice and report the total CPU and elapsed time spent in ‘B’, but these numbers do not include the time spent in the various invocations of ‘C’. This is different from the way CPU time was reported in a similar section in the existing Web container and EJB container SMF record 120 record subtypes.

Finally, the entry for ‘A’ contains CPU and elapsed time spent in ‘A’ and does not include the time spent in ‘B’ or in the direct and indirect calls to ‘C’.

If you have a common application that is called by several other applications, all within the same server, this breakdown can help you determine how much the common application is used by the other applications.

User data

In spite of our best efforts to think of every bit of information you might possibly want in the subtype 9 record, there is likely to be something you need that we didn't think of. Or, perhaps there is some information you would like in the record that WebSphere has no way of knowing (such as something internal to your application). For these reasons, we have added a user-data section to the subtype 9 record. At any point during the dispatch of a request, from the thread where the request was dispatched, you can call a WebSphere API and provide us with a block of data up to 2 kilobytes for inclusion in the subtype 9 record written for the dispatch of the request. The API is part of the **SMFEventInfrastructure** described in the WebSphere Information Center.

You can call this API from within your application or from something else such as a servlet filter. If your application runs on several platforms, the 'addData' API does nothing when called from an application running on another platform.

You might only want to bother with the overhead of building your block of data if subtype 9 records are being written. The **SMFEventInfrastructure** provides an API to query whether WebSphere is configured to write the records. Additionally you can register an SMFEventNotifier, which is driven when the subtype 9 records are turned on or off. Note that this notification and the query API only tells you whether WebSphere is building and calling the SMF API to write the records, other SMF configuration (such as the SMF parmlib member) can prevent the records from actually being kept.

Also remember that this data is accumulated in the servant region. If the request doesn't complete cleanly because the servant region was abended (perhaps for a timeout) the user data doesn't get copied into the controller to be written into the SMF record. A minor code in field **SM1209CJ** would indicate this situation.

However, suppose you have several applications that should add user data to their SMF records. How do you tell them apart so that you can format them properly when looking at the records? To help with this problem there is a 'type' associated with the user data section in field **SM1209FF**. IBM reserves type values less than 65535 for its own use. It is a four-byte field, however, which leaves you quite a few values for your own use. Your formatting code can examine the 'type' and determine how to format the data.

Because the triplet at the top of the record indicates the length of the section, all of the user data sections must be the same length, so that you can traverse the record and find them all. However, the length of the data in each blob could be very different. Therefore, there is also a length in the user data section in field **SM1209FG**. This field is the length of the data in the section; however, it should not be used to find the next section. Use the section length value in the triplet for that. Of course, if no user data was supplied, the triplet is set to zeroes.

What if two calls are made to set user data with the same identifier? The data kept for that identifier is replaced with the new data. It does not append the new data to the existing data, it replaces it.

How many user data sections can you have? We limit it to five unique types per subtype 9 record. We restrict the number to keep the overall size of the record under the 32K maximum SMF record size. The call to set user data into the record sets a return code to indicate whether you have replaced existing data or if your data was ignored because there are already too many user data sections.

© 2008,2011, IBM Software Group, Application and Integration Middleware Software, Poughkeepsie, NY

Configuration to Enable SMF Recording

SMF recording for the SMF 120-9 (and -10) are controlled by environment variables that can be set in the admin console. The values are printed at server startup, but can be modified dynamically (see the next section). The Knowledge Center describes the variables but never comes right out and says which variables control which records and sections.

server_SMF_request_activity_enabled – set to '1' tells WAS to try to write 120-9 records for request processing.

server_SMF_request_activity_CPU_detail – set to '1' tells WAS to include the CPU detail sections in the 120-9

server_SMF_request_activity_timestamps – set to '1' tells WAS to include the formatted timestamp section in the 120-9

server_SMF_request_activity_security – set to '1' tells WAS to include the security section in the 120-9

server_SMF_request_activity_async – set to '1' tells WAS to write 120-9 records for async work

server_SMF_outbound_enabled – set to '1' tells WAS to write SMF 120-10 records for WOLA outbound requests

Operational considerations

Unlike the existing record 120 subtypes, the subtype 9 record sports both a DISPLAY command and a set of MODIFY commands.

The MODIFY commands all begin with:

```
MODIFY <server>,SMF,REQUEST
```

following which, you can specify ON or OFF to turn the entire subtype 9 record on or off. With the record on, you can also turn on or off certain sections. For example, you might turn the CPU usage section off to avoid the overhead associated with collecting the data. Or, you might turn the formatted timestamps section off to avoid the increase in record size to contain the strings. Or, you might turn the security section off, to avoid the overhead associated with collecting the data and to reduce the record size.

Of course, if you want the data in these sections, configure them on. The MODIFY command lets you change the settings dynamically. Requests that are in flight when you enter the MODIFY command might not "see" the change in configuration.

The complete set of options for the command is:

```
MODIFY <server>,SMF,REQUEST,ON
MODIFY <server>,SMF,REQUEST,OFF
MODIFY <server>,SMF,REQUEST,CPU,ON
MODIFY <server>,SMF,REQUEST,CPU,OFF
MODIFY <server>,SMF,REQUEST,TIMESTAMPS,ON
MODIFY <server>,SMF,REQUEST,TIMESTAMPS,OFF
MODIFY <server>,SMF,REQUEST,SECURITY,ON
```

© 2008,2011, IBM Software Group, Application and Integration Middleware Software, Poughkeepsie, NY

```
MODIFY <server>,SMF,REQUEST,SECURITY,OFF
```

We have also added a new DISPLAY command to help you determine the state of recording for the subtype 9 record:

```
MODIFY <server>,DISPLAY,SMF
```

The resulting messages indicate whether the record itself is on or off, and whether the optional sections are on or off. The messages also indicate the number of SMF records that were written and the time when the last write occurred.

If you are having trouble gathering the subtype 9 record, we will also show you the number of failed attempts to write the record and the timestamp of the last failure. For a failure, we will also show you the last non-zero return code from an attempt to write an SMF record. This return code can tell you, for example, that the record was not written because an SMF exit you have installed decided to suppress the record (all SMF records are given to installation exits before they are written to the SMF data set).

The SMFWTM macro is used to write records to the SMF data set. For a description of the SMFWTM return codes, see the book *z/OS MVS System Management Facilities (SMF)*, SA22-7630.

Here is an example of the output from the DISPLAY command:

```
BBOO0344I SMF 120-9: ON, CPU USAGE: OFF, TIMESTAMPS: OFF, SECURITY
INFO: OFF
BBOO0345I SMF 120-9: TIME OF LAST WRITE: 2008/05/31 15:32:51.112,
           SUCCESSFUL WRITES: 2366157,  FAILED WRITES: 0
BBOO0346I SMF 120-9: LAST FAILED WRITE TIME: NEVER, RC: 0
BBOO0188I END OF OUTPUT FOR COMMAND DISPLAY,SMF
```

SMF browser updates

Update – January 2022

In mid-2020 the SMF browser and the plugins described in WP102312 were moved to open source in github (https://github.com/IBM/IBM-Z-zOS/tree/main/SMF-Tools). The browser itself was broken into two .jar files (SMF_CORE and SMF_WAS). Usage documentation can be found in the ReadMe. Note that package names changed, so the command examples below won't work exactly as-shown. Going forward only the github browser will be updated. The 'old' one will stay where it is, for now anyway.

Although many of you already have existing programs and processes to handle SMF records, WebSphere provides a Java program to parse SMF records and produce a somewhat human-readable formatted report. The IBM Washington Systems Center has updated the browser to produce a summary report, which is now further enhanced to process the subtype 9 record.

While we were messing around in the SMF browser, we had a little time to play and added some features. However, the existing parameter parsing did not allow for much expansion and the rules say that we cannot rearrange the parameters because doing so might break programs that use the old parameters. Therefore, the existing way to invoke the browser still works and it produces a formatted report of all the records and the summary as before, but it now includes the subtype 9 records.

But there's also a new way to invoke the browser. We created a new 'main' program inside the .jar file. The class to invoke is **com.ibm.ws390.sm.smfview.SMF**. The new main program takes two input parameters: INFILE and PLUGIN.

The INFILE parameter is followed by the name of the data set containing the SMF data to be parsed, like this: `INFILE(HUTCH.SMFDATA.SYSB.D080822)`. The PLUGIN keyword is optional. If omitted,

the browser produces the formatted report just like before, without output directed to stdout. If you want to redirect the output, you can specify `PLUGIN(DEFAULT,filename)`. Specifying STDOUT for the filename directs the output to standard out.

If you want the summary report instead, specify `PLUGIN(PERFSUM,filename)`. Again specifying STDOUT for the filename sends the results to standard out.

Example: An invocation might look like this (all on one line of course):

```
java com.ibm.ws390.sm.smfview.SMF  "INFILE(HUTCH.SMFDATA.SYSB.D080822)"
"PLUGIN(PERFSUM,/u/hutch/smf.txt)"
```

If you'd like to write your own plugin, you can — the source is included in the .jar file. You can do so by implementing the **com.ibm.ws390.sm.smfview.SMFFilter** interface. You cause your plugin to get control by specifying its class name instead of DEFAULT as the first part of the PLUGIN parameter, like this:

```
java com.ibm.ws390.sm.smfview.SMF  "INFILE(HUTCH.SMFDATA.SYSB.D080822)"
"PLUGIN(com.mycompany.smf.myplugin,/u/hutch/smf.txt)"
```

Your class will need to provide five methods to implement the interface, as follows:

- The **initialize** method, which gets control during initialization and is passed the string that is the second part of the PLUGIN parameter.
- The **preParse** method, which gets control after each record is read and can examine the record type and subtype to determine if processing should continue for this record. This method returns a Boolean value indicating whether parsing should proceed.
- The **parse** method, which is called to do the actual parsing. If you just want to let the existing code handle it, call the **commonParse** method in the DefaultFilter class. If you want to handle some other record type besides record 120, this is the place to put the code.
- The **processRecord** method, which is called after parsing is complete. This is where you can do any post-processing by accessing the various sections of the parsed SMF record.
- The **processingComplete** method, which is called after all the records in the SMF data set have been examined. Your final report can be written from here.

As an example, we have implemented the **ClassificationXMLFilter class**. It examines any subtype 9 records in the data set and extracts the classification data for HTTP and IIOP requests. The class uses that data to produce an outline of the XML file used to determine the transaction class provided to WLM when requests are classified.

To invoke this filter, run the browser like this:

```
java com.ibm.ws390.sm.smfview.SMF  "INFILE(HUTCH.SMFDATA.SYSB.D080822)"
"PLUGIN(com.ibm.ws390.sm.smfview.ClassificationXMLFilter,/u/hutch/smf.txt)"
```

Results go to the specified output file. If you don't have a classification XML file and are thinking of creating one, the output from this plugin might provide a starting point.

Understand that the browser is provided as-is and is not part of the WebSphere Application Server product. If you have problems or complaints or suggestions, we are happy to listen, but might not be able to do anything about it.

Update: 7/11/2013

The browser has been updated with some properties to help thin out the data being processed by the browser for cases where your SMF data includes records from a lot of different servers. The properties are documented with the browser, but I'll list them here too, in case more of you read this paper than the actual tool documentation…

If you are processing SMF 120-9 records there are three properties that are available to limit the output. If you have SMF data from more than one z/OS system you can specify the system name you are interested in and only 120-9 records from that system will be processed. Note that plugins will get control for the pre-parse and parse phase, but not the process phase. The property name is:

`com.ibm.ws390.smf.smf1209.MatchSystem`

If you have SMF data from more than one server, you can specify the short server name of the server you are interested in using this property:

`com.ibm.ws390.smf.smf1209.MatchServer`

And finally, if you would prefer to ignore SMF 120-9 records written for 'internal' requests (those with request types of Unknown, Mbean, OTS, or Other) specify a value of 'true' for this property:

`com.ibm.ws390.smf.smf1209.ExcludeInternal`

Here is an example of invoking the browser to get the default report but just for a server named ABCXYZ, excluding records for internal request types.

```
java -cp bbomsmfv.jar -Dcom.ibm.ws390.smf.smf1209.MatchServer=ABCXYZ
-Dcom.ibm.ws390.smf.smf1209.ExcludeInternal=true
com.ibm.ws390.sm.smfview.SMF "INFILE(USER.SMFDATA)"
```

Interval records

While the existing record 120 subtypes can be written per request or consolidated into an interval record, the new subtype 9 record is written only per request. There is no corresponding interval record in this release. It's difficult to determine what the most useful content of an interval record would be in this case. Simply combining all of the data accumulated across possibly thousands of request records into a giant multi-part interval record does not seem useful, nor could we decide how to accumulate data into a consolidated interval record in a way that would be helpful to most installations.

We have included this section in this document for two reasons. The first is to explain why there isn't a matching interval record for the new request record. The second is to request your input. If you are interested in consolidating the data from the new subtype 9 record into an interval record, how would you want it done?

There is no guarantee that we will ever add an interval record or that we'll be able to respond to or incorporate your suggestions. However, if you have suggestions about how to design a matching interval record, feel free to send me a note (follis@us.ibm.com).

Thanks.

Conclusion

We hope that the new subtype 9 record contains all of the information you need to do chargeback and capacity planning for WebSphere based applications. If not, you can use the user data section in the subtype 9 record to add whatever information you need.

The MVS MODIFY command is enhanced to allow you to turn data collection for the record on and off dynamically, as well as choose to dynamically enable or disable certain sections of the record to improve performance or reduce the size of the record.

© 2008,2011, IBM Software Group, Application and Integration Middleware Software, Poughkeepsie, NY

Updates for Version 8!

In this ‘appendix’ we will take a look at the changes and enhancements made to the WebSphere Application Server SMF recording in Version 8.

Things That Moved Around or Changed

Before we get into all the new cool new things in Version 8 we should quickly cover a couple of minor changes that we made. The first change involves the SM1209CE field. This field was designed to contain the string that indicates the build level of the server code. The field is eight characters long.

Unfortunately, our ever-evolving build process didn’t know we had an eight character limit on the length of the build string and began to generate build-level names that didn’t fit. To address this we created a new field, SM1209HV, which is 16 characters long. Hopefully that will be enough. Fortunately there was some reserved space right after SM1209CE so we just wrapped SM1209HV around the old field and tacked-on eight bytes taken from the reserved space. This means the old field is really just the first eight bytes of the new field.

The other change was just a slight reorganization. We have a limit on the number of CPU Usage sections we can fit in the record. If we overflow that limit we needed a bit to tell you there were not as many of these sections as there should have been. Since we already had a pile of bits in SM1209DT in the Request Information Section we just put the overflow bit (SM1209DY) there. But it really does not go there. It is actually information about the record, not the request. So in Version 8 we moved this to bit SM1209GF in the Server Information Section.

We had a similar problem with bit SM1209FJ which tells you whether we got GMT offset information from a call to CEEGMTO. Again, we put the bit in the Request Information Section because we had a convenient pile of bits there. But it really does not go there. So in Version 8 we also moved this bit to the Server Information Section in bit SM1209GG.

Both of the original bits (SM1209DY and SM1209FJ) remain and can be used, but they are deprecated and you should change any code that cares to examine the new bits (SM1209GF and SM1209GG).

We also added more formal SMF support for WebSphere Optimized Local Adapters (WOLA). For WOLA requests inbound to the server the SM1209CK field (request type) gets a new value of ‘12’ (decimal) to indicate this is a WOLA request. Prior to Version 8 WOLA requests got marked as IIOP due to similarities in the path through the code. We also updated the classification information section for WOLA. There are two new values for SM1209EM to indicate the WOLA service name or the CICS imported transaction name. If the transaction name was used in classification with WLM then the SM1209FQ bit will be set.

Affinity Routing Information

In Version 8 we expanded a little on the information we provided with a single bit in the SMF 120-9 record. Which bit are we talking about? The field is called SM1209DZ. We set that bit in the 120-9 record when a request was routed to a particular servant region because of an affinity.

© 2008,2011, IBM Software Group, Application and Integration Middleware Software, Poughkeepsie, NY

What does that mean? Well, suppose an HTTP request comes into the server. This is a new guy; we've never seen him before. So his request is allowed to run in any servant that is part of this server. And it does. As the request executes it creates an HTTPSession object. This is something like a shopping cart for this user. It is an object in the memory of this servant region. When another request from the same user comes in the application that runs is going to want to find this data to remember what it was doing. For example, if he clicks 'buy' you want to know what stuff he decided to buy. Customers hate it when you just guess.

Therefore, when his 'buy' request comes in we have to be sure that the request runs in the same servant region that the original request ran in and created the session object. To make that work we tuck something special in the response we send back to the original request. On subsequent requests the client (the browser probably) sends back that same special thing. We use that to find the right servant region. For those who like to dig under the covers, we do this with a z/OS GRS ENQ and the special thing we send back includes the ENQ RNAME.

But we should get back to the SMF record. What does that SM1209DZ bit mean? We set that bit when we place a request in a servant region because it has affinity to that servant region. In our example that would mean that the 'buy' request would be queued with affinity. The first request that created his shopping cart defined the affinity, but would not have the SM1209DZ bit set because it was allowed to run anywhere.

This lets you look at your SMF data and see how many requests ran in each servant region because WLM decided to run it there and how many ran in each servant region because we had no choice.

But what you can not do is tie the requests together. You can not tell that 'this' request that WLM was allowed to route created an affinity that was later used by 10 requests, or only 1, or never. To help answer these questions we added a few fields to the z/OS Request Information Section of the 120-9 record. The version number of this section (SM1209CL) was incremented to allow you to tell if the data is there or not.

The first pair of fields we added are SM1209GH and SM1209GI. These fields are set to the length and value of the special string we used for the ENQ if this request created an affinity. Thus a non-zero value in SM1209GH tells you that this request created an affinity.

The next pair of added fields are SM1209GJ and SM1209GK. If the SM1209DZ (affinity) bit is set, then these two may be set to indicate what string was used to find the right servant region. It is possible to have SM1209DZ set and not have an affinity string. There are cases, especially MBeans, where a request is deliberately routed to all the servant regions and thus the requests are queued with affinity to a servant but it is not because of a previous request.

But in more normal cases you should be able to take the string in SM1209GK and match it to a string that appeared in a previous request record in SM1209GI. That will let you construct the chain of SMF 120-9 records for this affinity. You can then calculate things like the average number of subsequent requests that come from an established affinity. Affinities mess up WLMs ability to balance work between the servant regions. The more requests in an affinity-chain the less you are allowing WLM to control things.

You might also be able to notice some requests that establish an affinity that no subsequent request uses. These might represent 'abandoned' shopping carts, which could be interesting to know about. Or possibly a problem in the application because it is unnecessarily creating an HTTPSession that is never used. Or it might be something else entirely....

Hopefully these new pieces of data about requests will let you find out something you did not know about your environment.

Granular RAS Control Information

A major z/OS-only improvement in Version 8 came from the Granular RAS Controls added to the classification XML file. We won't go into all the details about this feature here, but it is important to note that the SMF 120-9 data can be used to verify that the classification XML is assigning attributes to your application requests the way you want. All the various XML-assigned attributes are replicated in the updated SMF120-9 record in the Request Information Section. Some of these attributes are really only useful to have to validate the XML is working as expected, but others might have value on their own. Let's take a look.

First we have some more bit flags. The first of these is SM1209FK. If this bit is set then classification trace is turned on for this request. Put simply, this feature allows you to collect the configured trace only for requests that match the pattern specified in the classification XML. This bit tells you if this particular request matched an XML entry with the classification_only_trace flag turned on or off.

The XML file also allows you to control whether you get SMF 120-9 records for any particular request, as well as what optional sections you get. For example, you might want to turn off the 120-9 records for internally generated requests and turn on the CPU usage detail for a particular web application. The XML file provides that sort of granular control. The SMF record for each request tells you which options applied to the request. The optional sub-sections are indicated by SM1209FN, SM1209FO, and SM1209FP. Whether the record itself is configured on or not is indicated by SM1209FM.

At this point you might be asking, "But can't I tell by looking at the record, if I even have a record, what options were set?" It sure seems like you should never see a record with SM1209FM turned off. If writing the record is off, how did you get the record? The answer lies in the fact that these bits just record what the XML file indicated the server should do with this request. It is possible to override the XML file dynamically using MVS Console Modify commands. Therefore these bits are best used to validate your XML file is behaving as expected and not to determine what the SMF record is supposed to look like.

The next bit, SM1209FQ, relates to WebSphere Optimized Local Adapters (WOLA) and WLM classification as discussed earlier.

The next several fields were intended to help understand what timeout properties applied to this request. In previous releases you could just look at the server configuration and see, for example, what dispatch timeout value applied to all HTTP requests. With timeout options configurable in the classification XML file it becomes possible for different requests to get different timeout options. While this adds a lot of power and flexibility, it can be difficult to figure out what timeout value applied to a particular request. To help with that, we added the various timeout option values used for the request into the SMF 120-9 record.

The values are:

SM1209FR	Documentation to gather for a timeout (callstack, etc)
SM1209FS	Documentation to gather for a CPU timeout
SM1209FT	Documentation to gather for a Dispatch Progress Monitor event (DPM)
SM1209FU	Timeout Recovery option for this request (SESSION/SERVANT)
SM1209FV	The dispatch timeout used for this request
SM1209FW	The actual queue timeout value (configured as a percentage of the whole timeout)
SM1209FX	The timeout for outbound IIOP requests made under dispatch of this request

SM1209FY	The CPU time limit for the request
SM1209FZ	The Dispatch Progress Monitor interval

The final field introduced into the SMF 120-9 record for Granular RAS Controls is the message tag. The purpose of this tag in the XML file was to allow you to define an eight character 'tag' to associate with a set of requests. This tag would then be placed in any messages or traces issued while the request was in dispatch. One use of this might be to help determine which of several applications installed in a single server was the cause of some particular message.

To help validate that your XML is working properly, we also included the message tag in the SMF 120-9 record. However, this tag might also help you process your SMF records by grouping records written for requests to the same application (or set of applications) together.

SMF Type 120 Subtype 9 Async Work Records

There are a number of ways to run work asynchronously inside a WebSphere Application Server. For example, you can use Asynchronous Beans. In Version 8 Fix Pack 1 (8.0.0.1) you can exploit Asynchronous Servlet support or Asynchronous Session Beans. Ultimately all of these, among other options, usually make use of a Work Manager to run the asynchronous work.

A Work Manager maintains a pool of threads that it uses to run asynchronous work. In Version 8 we added support for SMF recording of asynchronous work. Because the SMF 120-9 record records work done by the server we choose to extend the subtype 9 record to also be used to report on asynchronous work.

How can you tell if an SMF 120-9 record is for 'normal' work or asynchronous work? Records written because the server dispatched some type of inbound request will include the Platform Neutral and z/OS Request Information Sections (SM1209AN and SM1209AQ will be set to one). Records written for asynchronous work will not have these sections (those fields will be zero) but will have the new Async Work section (new field SM1209GD will be set to one). Asynchronous work records may also have CPU usage sections (SM1209BF is greater than zero) or User Data sections (SM1209FD is greater than zero). The Server Information sections (Platform Neutral and z/OS) will always be present regardless of what type of work the SMF 120-9 record is recording.

So what is in this new Asynchronous Work section? After the mandatory section version number we have a few time stamps. These are 16 byte STCKE (store-clock-extended) timestamps. We include the time the execution context was created (SM1209GL) which you might not have if the application didn't create an execution context. This is followed by the more-interesting execution start and end times (SM1209GN and SM1209GO). Subtracting these gives you elapsed time to run the asynchronous work. Remember to use the GMT offset earlier in the record if you need to adjust these to local time.

After the timestamps we identify the servant region where the asynchronous work ran. We provide the servant Process ID (PID) in SM1209GP, the jobname (SM1209GQ), the jobid (SM1209GR) which is the STCxxxxx name, the stoken (SM1209GS), and the ASID (SM1209GT). The jobid is probably the easiest thing to use to correlate records for work that ran in the same servant, although the stoken will do that as well.

If an execution context was created, it would be good to know who did it. For asynchronous work created from a dispatched request, this helps you tie this SMF record to the 120-9 record for the request that did the scheduling. We include the USS Task ID (SM1209GU), the TCB address (SM1209GV), and the TTOKEN (SM1209GW).

After that you would want to know about the thread that ran the asynchronous work. This can help you determine how many threads in the work manager pool are being used to run the work. As with the execution context thread, we give you the USS Task ID (SM1209GX), the TCB address (SM1209GY) and the TTOKEN (SM1209GZ).

The next section of information relates to WLM enclaves. Asynchronous work can run under the same enclave as the dispatched request that scheduled it or it can create its own enclave. If an enclave is created there are different transaction class names that can be used to create it. If we picked up an enclave from the execution context of the thing that scheduled the work it will be in SM1209HA. This is probably the easiest thing to use to correlate this record with an SMF 120-9 record for a 'normal' request that scheduled asynchronous work.

The enclave token used to dispatch the request is reported in SM1209HB. This might be the same as the enclave token in SM1209HA or it might not. If we created a new enclave, the transaction class name used to help WLM classify the work is reported in SM12109HC. How this classification works is rather complicated and hopefully reporting this here will help resolve issues where asynchronous work does not get classified as expected. The last bit of enclave related information is the first of our two bits flags. SM1209HE is a bit that shows whether the dispatch enclave was created new (1) or if we used the enclave provided in the execution context created to run this work (0).

The other bit flag indicates whether the work was scheduled with the 'is-daemon' flag set or not. Daemon work creates a new enclave a little differently and also runs on a newly created thread instead of using a thread from the work manager pool. The flag value matches the value of the Boolean parameter, so is-daemon set to 'true' maps to a '1' in SM1209HF.

We also include some CPU time measurements. First we ask WLM about CPU time used by the enclave associated with the thread where the asynchronous work is dispatched. We use the IWMEQTME API to get the information. Remember that this might be a new enclave created just to run this asynchronous work or it might be the same enclave used to dispatch a 'normal' request. It is also possible that the enclave was propagated to another local WAS server over an IIOP connection. WLM provides us with several values. We report it all in the SMF record.

There was some confusion about which value was which and we took a doc APAR (PM74221) to clean up the infoCenter. The descriptions in the mapping macro are very terse and its easy to get them confused, so use the infoCenter (the official doc) or this paper (less official). The 'HJ' and 'HL' values were originally described backwards in the infoCenter.

The values include: total CPU time used under the enclave (SM1209HG), just zAAP CPU time (SM1209HH), and zAAP eligible CPU time than ran on a general purpose CP (SM1209HI). This last field might be non-zero if you are configured to run zAAP offload but don't actually have a zAAP. We also report the CPU time that ran on a zIIP (zAAP on zIIP) in SM1209HJ, was qualified to run on a zIIP (SM1209HK) and the zAAP on zIIP eligible than ran on the general purpose CPU (SM1209HL). Finally, WLM provides us with a normalization factor for the zAAP CPU time in case the GPs and zAAPs are not running at the same speed (SM1209HM).

Be careful with the values returned by IWMEQTME. I've seen them sometimes be smaller than it seems they should because WLM is really just returning whatever values it finds associated with the enclave, which might not have been updated recently.

Because the enclave CPU times might include time spent on other threads or other processes we also just simply ask z/OS for the CPU time on the thread where the asynchronous work ran. We ask at the beginning and end of dispatch and subtract, reporting the results in SM1209HN and SM1209HO. The first field contains total CPU time and the second contains CPU time on non-standard CP types (zAAAPs and zIIPs).

We have given you a lot of good information about when and where the work ran and how much CPU resource it used, but we have not yet told you what it was that ran. The next several fields help with that. First up we provide the class name of the 'Work' object that ran. SM1209HP has the length of the

class name and SM1209HQ has the name itself (truncated to 128 bytes if necessary). The class name includes the package name which might make it long enough to get truncated.

Since you can configure more than one work manager to run asynchronous work we also tell you the name of the work manager. This is important if you are trying to use these SMF records to monitor thread usage within a work manager thread pool. The length of the work manager name is in SM1209HR and the name itself is in SM1209HS.

The only thing left to include in the record is who did the work. The SM1209HU field contains the user identity that was placed on the thread when the work was executed. The SM1209HT field tells you the length of the string in SM1209HU. The identity string will be truncated to 64 characters if necessary.

What about operational control of these new records? How do you turn them on and off? How can you tell if the records are being written? Since these are really sort of an extension of the SMF 120-9 record introduced in version 7 we treat it as if it were an optional section in the records written for normal requests. To get SMF records reporting on asynchronous work you need to have the base request record turned on, either by setting server_SMF_request_activity_enabled or by using the MVS Console Modify command:

```
MODIFY server,SMF,REQUEST,ON
```

Then you can choose to also enable recording for asynchronous work by setting the server_SMF_request_activity_async environment variable to ‘1’. You can also turn them on and off dynamically with the following MVS Console Modify commands:

```
MODIFY server,SMF,REQUEST,ASYNC,ON
MODIFY server,SMF,REQUEST,ASYNC,OFF
```

SMF recording enabled or disabled in the classification XML has no effect on whether SMF 120-9 records are written for asynchronous work. The ASYNC records only consider the global setting done via environment variable or the Modify command.

The Display command has also been enhanced to report on the configuration and results of any attempts to write SMF records for asynchronous work.

Version 3 of the SMF Type 120 Subtype 9 z/OS Server Info Section.

In 2019, in 8.5.5.16 and 9.0.0.11 a couple of small changes were made to the SMF 120 Subtype 9 record. These changes were in the z/OS Server Information Section. The length of the section did not change, but the version of the section was changed to indicate that previously reserved fields now contain data.

The first change was an update to the SM1209GE field which contains flags. A new flag, SM1209HW, was introduced. This flag reflects the state of the CVTZCBP flag on the system where the record was written. See the z/OS documentation for that CVT bit for details.

The second change was the introduction of the SM1209HX field into part of a formerly reserved field in the z/OS Server Information Section. This field contains the number of application worker threads

© 2008,2011, IBM Software Group, Application and Integration Middleware Software, Poughkeepsie, NY

defined for the servant regions in this server. This value might be calculated based on server configuration (the “workload profile” setting of IOBOUND or CPUBOUND, etc) or set to a specific value from the configuration (when the workload profile is “CUSTOM”). From the 120-9 records you can tell how many of the worker threads are processing work (by keeping track of all the TCB addresses that are used to dispatch requests), but until now you couldn’t tell how many threads there actually were.

New SMF Type 120 Subtype 10 – Outbound Requests – WOLA (8.0.0.1)

We already mentioned some changes to the subtype 9 record in support of WebSphere Optimized Local Adapters (WOLA). But the first maintenance drop of version 8 (8.0.0.1) contains some major new SMF function for WOLA. We added a whole new SMF Type 120 record subtype!

The all-new SMF 120 Subtype 10 record is intended to describe outbound requests made by an application running inside a WAS Servant Region. At present the only type of outbound request we support is a request made over WOLA. Will we someday support other outbound request types? Maybe. Maybe not. If you have requirements, get ‘em open.

What is in the subtype 10? Well, it starts off looking a lot like the subtype 9. We have the exact same Platform Neutral and z/OS-only Server Information sections. These two sections identify the server that is writing the record and, again, look exactly like these sections in the subtype 9. Hopefully that will make processing of the two record subtypes easier.

In the following sections of this paper we will go through all the other sections of the subtype 10 that are unique to that subtype.

Outbound Request Information

This section will be present in all SMF Type 120 Subtype 10 records. It provides information common to all outbound requests regardless of type. It begins, of course, with a version number in SM120ACF. Remember that the subtype 9 record fields all started with SM1209. The subtype 10 records start with SM120A (because 0xA is 10 in decimal).

The first question to answer with this record is where the request came from. For that information you can turn to SM120ACG for the USS Process identifier (PID) of the originating servant region. Remember that the earlier two sections (common with the subtype 9) point to the server, but do not specify a particular servant region. That information is in this section.

In addition to the PID we also have the jobname, jobid, ASID, and STOKEN of the servant. These are in fields SM120ACR, SM120ACS, SM120ACT, and SM120ACU respectively. But which task within that address space? We provide fields SM120ACH, SM120ACV, and SM120ACW to give you the USS Task ID, the TCB address, and the TTOKEN. Remember that asids and TCB addresses get reused. A STOKEN and TTOKEN are unique within the IPL.

The servant and task identifiers might help you correlate this outbound request with a dispatched inbound request that sent it, but there is an easier way. Remember that every request dispatched inside a servant region (and some asynchronous work) has a WLM enclave token associated with it. That token appears in the Type 120 Subtype 9 record written for the dispatched request (including asynchronous work). The SM120ACY field will contain the enclave token associated with the thread

© 2008,2011, IBM Software Group, Application and Integration Middleware Software, Poughkeepsie, NY

making the outbound request. This should allow you to correlate a set of outbound request subtype 10 records with the subtype 9 record of the originating dispatched request.

Outbound requests usually contain some data and the response might as well. Sometimes it is useful to know much data was sent and received. The SM120AD1 and SM120AD2 fields contain the count of bytes sent and received in the outbound request/response for this subtype 10 record.

Measuring response time is always important. The subtype 9 record will provide the timestamps for the beginning and end of the dispatch of a request. But it is possible a lot of that time might have been spent waiting for a response to an outbound request directed to something outside of this server. It would be nice to know how much of the 'dispatch time' was actually spent waiting for something somewhere else. To help with that we provide SM120AD3 and SM120AD4 which are the STCK-format timestamps indicating when the outbound request was sent and when a response was received.

That concludes the 'basic' information about the outbound request. For more details we need to know what kind of an outbound request this was. To find that out you look at field SM120ACK. If this field is a '1' then this is a WOLA outbound request. If you use the WOLA APIs to drive a request to IMS over OTMA then the SM120ACK field will contain a '2'.

Now we will take a look at the optional additional sections you might get depending on what type of an outbound request this was.

WOLA Outbound Request Information

If this was a WOLA outbound request (SM120ACK=1) then you will get the WOLA outbound request information section. It begins, of course, with a version number in SM120AD5. That is followed by the WOLA registration name of the target. Remember that something outside of WAS acting as a WOLA server will use the WOLA 'register' API to identify itself to WAS as a target of outbound-from-WAS WOLA requests. Running inside of WAS, the application (or connection factory) knows the registration name of the server it wants to target. That name goes in the SM120AD6 field.

In addition to registering with WOLA the target can also supply a 'service' name. This allows a single process to 'serve' multiple different types of requests coming out of WAS. The application knows the service name it requires and specifies it on the WOLA API to make the outbound request. This service name is given to the target along with the request body. The name is also included in the subtype 10 record in SM120AD6. These two fields essentially tell you what the outbound request was for.

The final field in this section (SM120AD8) only applies when the target of the outbound request is CICS. WebSphere generates a unique identifier for this request. We put the identifier in the subtype 10 record. WOLA also sends it to CICS as part of the request. If you are running CICS TS 4.2 it is possible to have this identifier also included in the CICS SMF 110 record. For this to work you have to run a new link invocation task (BBO#) for each transaction. This is the default behavior unless you specified REUSE=NO when you start the link server (BBO$).

This correlator will show up in the transaction resource monitoring section in the OADID and OADATAx fields. This should allow you to build a one-to-one correlation between SMF 120-10 records and CICS 110 records. You can then use the enclave token to correlate SMF 120-10 records with 120-9 records to get end-to-end record correlation.

Here is an example of the output from the CICS SMF report utility DFH$MOLS (just the part with the WOLA correlation data):

```
DFHCICS  C351     OADID             C9C47EE6 8582E297 88859985 40C19797 93898381   ID=WebSphere Applica
                           +X0014   A3899695 40E28599 A5859940 86969940 A961D6E2   tion Server for z/OS
                           +X0028   40404040 40404040 40404040 40404040 40404040
                           +X003C   40404040
DFHCICS  C352     OADATA1           C3C5D3D3 7EE6C1E2 F0F04040 40D5D6C4 C57ED5C4   CELL=WAS00   NODE=ND
                           +X0014   D5F14040 4040C3D3 E4E2E3C5 D97EC2C2 D6C3F0F0   N1    CLUSTER=BBOC00
                           +X0028   F140E2C5 D9E5C5D9 7EC2C2D6 E2F0F0F1 40404040   1 SERVER=BBOS001
```

```
                                +X003C  40404040
DFHCICS  C353      OADATA2              C9D5E2E3 C1D5C3C5 7E000001 28000000 02000000    INSTANCE=
                                +X0014  3D006C5E 8800C81E C185F433 AC000000 00011900      %;h H Ae4
                                +X0028  01000000 00000000 00000000 00000000 00000000
```

The OADID field identifies this data as coming from WebSphere. The DATA1 field identifies the server that originated the request. The DATA2 field is an request instance identifier which is just a hex value that will uniquely identify the request among other requests from this server (any servant). These three fields are concatenated together and reported in the SMF 120-10 record in the 256 byte field SM120AD8.

Outbound Request Transaction Information

If transaction information was propagated over the outbound request this section will be present. After the section version number (SM120ADA) you will find the WOLA transaction XID value in SM120ADB. When the WAS application is participating in a two-phase commit transaction with CICS it uses the XA transaction protocols to do this. There will, therefore, be an XA Transaction Identifier associated with the transaction. That is the XID and it gets passed to CICS to help them track the transaction.

Outbound requests to the same target that are part of the same XA transaction will have the same XID. So if your application begins and commits several transactions you can use the XID to tell which outbound requests were included in which transaction. Note that requests to different targets under the same transaction will also have different XIDs because they are different branches of the same transaction.

Outbound Request Security Information

If security information is propagated over the outbound request, the eight byte MVS userid will be found in this section in field SM120ADE. This is the eight byte SAF identity associated with the request. It will be carried into CICS, although depending on how the CICS side is handling inbound requests it might not be used.

WOLA Outbound to CICS Information

If the target of the outbound request is hosted inside of CICS then this section will be present. This section contains context that is passed along to CICS to help run the request. It is written to SMF as a single 80 byte block, but you can break it down into several pieces.

First there is an eyecatcher that should be “BBOAUCIC” in EBCDIC. That's eight bytes long which is followed by a two-byte version field, currently set to one. We then skip two bytes.

Twelve bytes in we have a four-byte flag field. So far we've only used the right-most three bits of the flag field to indicate (left-to-right) use Channel, use COMMAREA, and use Container. These three bits are mutually exclusive and tell you how to interpret fields later in the block.

Right after the flag word (offset 0x10) is a four byte transaction id. This is filled in if the application overrides the four-byte link transaction ID (either in the connection factory or the application).

What's next depends on the flag that was set earlier. If the COMMAREA flag is on, then nothing else is set in the block.

If the Container flag is set then the next sixteen bytes starting at 0x14 represent the name of the container. That brings us 36 bytes (0x24) into the block where we have a four byte container type (one for 'bit' and zero for 'character'). Then at 40 bytes into the block (0x28) we have the response container name and type, sixteen and four bytes in length respectively. The rest of the block is unset and reserved.

© 2008,2011, IBM Software Group, Application and Integration Middleware Software, Poughkeepsie, NY

However, if the Channel flag was set, then the sixteen bytes at offset 0x14 are the name of the channel used for both the request and response. The following four bytes at 0x24 represents the type of all containers within the channel. The rest of the block is unset and reserved.

WOLA Outbound OTMA Information

If you are using WOLA over OTMA to talk to IMS then you get this section. You can tell because SM120ACK will be a two. In this section you get information specific to the OTMA interface.

Since we are using OTMA you need to provide WOLA with the IMS group id, server name, and transaction name that will be given to OTMA to drive the request in IMS. This information is provided in fields SM120ADN, SM120ADO, and SM120ADM.

When you use WOLA to go outbound the API allows you to specify a register name and service name. For non-OTMA WOLA we included those in the WOLA Outbound Request Information section. We also include them in the WOLA OTMA section. However, they are not actually used for anything and might be left blank.

You could, however, establish a convention that applications using WOLA over OTMA set the 12 byte register name and 256 byte service name to something meaningful to you. These values will then show up in the SMF 120 Subtype 10 in the SM120ADK and SM120ADL fields. You could use these areas for whatever you like.

Operations

To enable writing of the SMF 120-10 records for outbound requests you can set the environment variable *server_SMF_outbound_enabled* to true (or '1'). You can also dynamically turn recording of outbound requests on and off. Use the following MODIFY command to turn recording on:

MODIFY server,SMF,OUTBOUND,ON

To force recording of outbound requests to stop, use this modify command:

MODIFY server,SMF,OUTBOUND,OFF

Finally, to go back to whatever the environment variable setting is you can enter:

MODIFY server,SMF,OUTBOUND,RESET

Note that there is no tag in the classification XML file to control whether outbound requests from a dispatched request are recorded.

The DISPLAY,SMF command has also been updated to report on the current setting for outbound request SMF records, how many we have written, and whether any errors have been encountered writing the records.

© 2008,2011, IBM Software Group, Application and Integration Middleware Software, Poughkeepsie, NY

Document change history

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

October 12, 2008	Original Version
September 8, 2010	Updated Security section
January 21, 2011	Corrected SM1209CI to microseconds
May 10, 2011	Corrected explanation of SM1209CX, contains GP time only, not zAAP/zIIP time only.
May 18,2011	Undo correction of SM209CX…I was right the first time. Sigh…
July 14, 2011	Added Version 8 information
March 5, 2013	Updated Async Work Enclave CPU descriptions in support of PM74221.
July 11,2013	Added text about the new properties supported by the SMF browser
June 3, 2014	Improved contents/offsets in 120.10 CICS outbound context
September 21, 2015	Corrected SM1209ACY → SM120ACY
April 1, 2019	Added information about configuring WAS to write the records. Also added information about SM1209HW and SM1209HX.
01/12/21	Modified SMF Browser Updates section to point to new location for browser/plugins in github.

End of WP101342

© 2008,2011, IBM Software Group, Application and Integration Middleware Software, Poughkeepsie, NY