WebSphere Application Server for z/OS V8

Format SMF120-9 User Data Sections with the SMF Browser using Custom Formatters

This document can be found on the Web at:
www.ibm.com/support/techdocs
Search for document number **WP101726** under the category of "White Papers"

Version Date: March 31, 2016
See Document change history on page 7 for a description of the changes in this version of the document

IBM Software Group
Application and Integration Middleware Software

Robert G Alderman
IBM Poughkeepsie
845-435-9418
ralder@us.ibm.com

Introduction

This techdoc describes how to format the user data sections of WebSphere Application Server (WAS) for z/OS SMF 120 subtype 9 records, using custom formatters that can be added on to the SMF Browser for WAS on z/OS.

As of WAS for z/OS v7, user applications are able to write their own data into SMF 120 subtype 9 records. SMF120-9 records contain "user data sections", where user applications can write data (up to 2KB) during a request dispatch. The user data is recorded to SMF with the rest of the SMF120-9 record.

The SMF Browser for WAS on z/OS reads and formats SMF 120 data, including SMF 120-9 records. However, since the content and format of a user data section is completely defined by the user application that wrote it, the SMF browser is unable to format the user data in any meaningful way, since the structure of the data is unknown. Instead, the browser simply dumps the data in raw hex format.

In order to better format user data sections, the user may define and add a custom formatter to the SMF browser.

Defining a SMF120-9 user data custom formatter

A user data custom formatter is simply a Java class, written to format user data sections of a specific type. The user data custom formatter must be defined with the following class name:

```
com.ibm.ws390.smf.formatters.SMFType120SubType9UserDataTypeXXX
```

where **XXX** is the type ID (in decimal) of the user data section. Each user data section has an associated type ID. The type ID is what was specified when the user data section was written:

```
SmfEventInfrastructure.addDataToSMF120Subtype9Record(int type, byte[] data)
```

For example, if your application writes user data with type ID 70000, then the custom formatter would be named:

```
com.ibm.ws390.smf.formatters.SMFType120SubType9UserDataType70000
```

The class must extend com.ibm.ws390.sm.smfview.UserDataSection. It must define a constructor that takes a single UserDataSection argument. It is recommended that this constructor call super(UserDataSection), which will populate the parent object's public fields (these fields are inherited by the custom formatter). Note: if the constructor calls super(UserDataSection), it will have to handle or throw UnsupportedVersionException and UnsupportedEncodingException. See the following code excerpt:

```
import com.ibm.ws390.sm.smfview.UserDataSection;
import com.ibm.ws390.sm.smfview.UnsupportedVersionException;
import java.io.UnsupportedEncodingException;

public class SMFType120SubType9UserDataTypeXXX extends UserDataSection
{
  public SMFType120SubType9UserDataTypeXXX(UserDataSection uds)
    throws UnsupportedVersionException, UnsupportedEncodingException
  {
    super(uds);
  }
 …
```

The custom formatter inherits the following fields from UserDataSection:

```
/** data type for this section (less than 65535 reserved for IBM use) */
public int m_dataType;

/** length of data in this section */
public int m_dataLength;

/** user data itself */
public byte m_data[];
```

The m_data field contains the user data.

UserDataSection defines a method named “dump”. This method is invoked by the SMF browser when formatting the user data section. The custom formatter must override this method in order to gain control and format the user data.

The dump method takes two arguments: an SMFPrintStream and an int. The int contains the triplet number for the user data section. For all intents and purposes the triplet number can be ignored. The SMFPrintStream is the stream to which the formatted data is written.

```
import com.ibm.ws390.sm.smfview.SmfPrintStream;
…
public void dump(SmfPrintStream aPrintStream, int aTripletNumber)
{
 …
}
```

SmfPrintStream contains many useful utility functions for formatting data. Please refer to the javadoc included with the SMF browser for more details.

Compiling and running the user data custom formatter

In order to compile the custom formatter, you must include the SMF browser jar (bbomsmfv.jar) on the classpath.

Once the custom formatter is compiled, the user must simply add the class to the classpath when invoking the SMF browser. The SMF browser will find and load it dynamically. No other configuration is required.

When the SMF browser encounters a user data section, it reads the type ID and searches the classpath for a custom formatter for that type, using the class naming pattern noted above.

If a custom formatter is found on the classpath, the SMF browser dynamically loads the class, instantiates an instance, passes the raw data to the instance, then invokes the instance to perform formatting.

If no custom formatter is found, the SMF browser formats the data using the default formatter, which simply dumps the data in raw hex format.

Example user data custom formatter

Looking at an example is probably the easiest way to understand how to define a custom formatter. Imagine a user application that writes SMF 120-9 user data with type ID 70000. The user data is simply a String. Here's what the custom formatter would look like:

```
package com.ibm.ws390.smf.formatters;

import com.ibm.ws390.sm.smfview.UserDataSection;
import com.ibm.ws390.sm.smfview.SmfPrintStream;
import com.ibm.ws390.sm.smfview.UnsupportedVersionException;
import java.io.UnsupportedEncodingException;

public class SMFType120SubType9UserDataType70000 extends UserDataSection
{

  public SMFType120SubType9UserDataType70000(UserDataSection uds)
    throws UnsupportedVersionException, UnsupportedEncodingException
  {
    super(uds);
  }

  public void dump(SmfPrintStream aPrintStream, int aTripletNumber)
  {
    aPrintStream.println("");
    aPrintStream.printKeyValue("Triplet #",aTripletNumber);
    aPrintStream.printlnKeyValue("Type","User Data for type 70000");

    aPrintStream.push();  // indents the following data

    String s = new String(m_data,0,m_dataLength); // read the string data

    aPrintStream.printlnKeyValue("The String",s);

    aPrintStream.pop();  // undo indent (outdent??)
  }
}
```

This custom formatter will generate the following in the SMF browser report:

```
Triplet #: 10; Type: User Data for type 70000;
  The String: whatever string was written by the user app
```

That's all there is to it. Happy formatting!

Appendix: Debugging class-load failures for the custom formatter class

If the custom formatter class is not being loaded, you can enable debug tracing by defining the following property. The debug trace will report any class-loading errors.

```
-Dcom.ibm.ws390.sm.smfview.UserDataSection.debug=true
```

Document change history

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

July 28, 2011	Original Version
March 31, 2016	First (finally) published

End of WP101726