

ALCS and WebSphere OLA

Using the WebSphere Optimized Local Adapter with ALCS

Dan Weber (daniw@us.ibm.com)
Senior IT Architect
IBM Software Group - Industry Solutions

V1.1 - 3 November 2010

Table of Contents

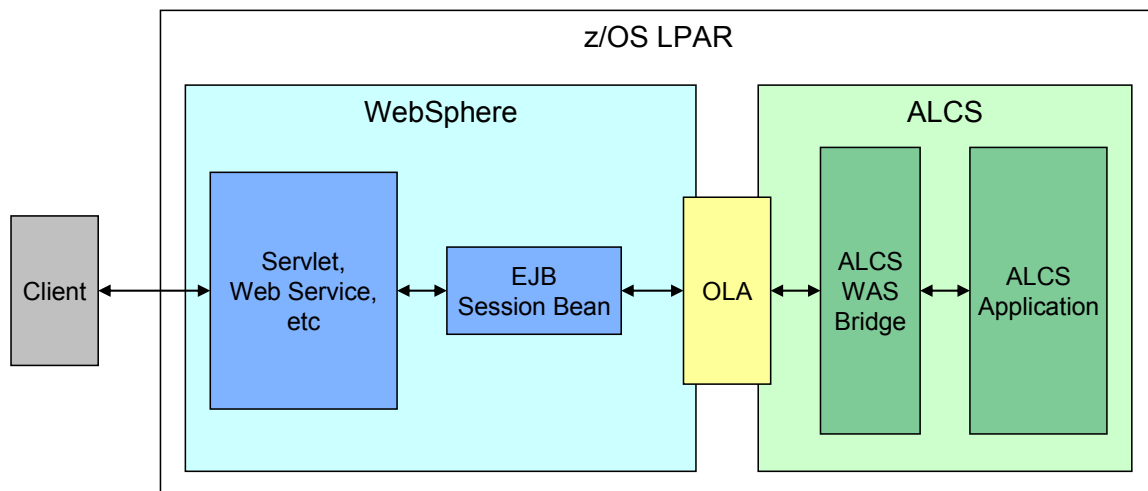
Introduction.....	3
Message Flow.....	4
Inbound Message Flow (WAS -> ALCS).....	4
Request Message Format	6
Response Message Format.....	6
Outbound Message Flow (ALCS -> WAS)	6
Outbound Message Format.....	7
Correlator.....	7
ALCS User Exits.....	8
Usage Scenarios	9
Web Services. Servlets, JSPs	9
MQ Series, JMS	10
EJB3 Considerations	10
Transaction Management.....	11
Prerequisites	12
ALCS	12
WebSphere.....	12
Sample Servlet Application	13
Overview	13
Message Format and Correlator.....	13
Code Listings	14
ALCS User Exit DXCUWAS3	14
ALCS User Exit DXCUWAS5	18
WebSphere Servlet ALCSServlet.....	22
WebSphere EJB Session Bean Interface ALCSConnection.....	25
WebSphere EJB Session Bean ALCSConnectionBean	27
Utility Class DataDumper	35
Sample Virtual Printer Application.....	36
Overview	36
Message Format and Correlator.....	37
Code Listings	37
ALCS User Exit DXCUWAS3	37
ALCS User Exit DXCUWAS5	37
WebSphere Servlet ALCSPrinterServlet	37
WebSphere EJB Session Bean Interface ALCSVirtualPrinter.....	39
WebSphere EJB Session Bean ALCSVirtualPrinterBean	40
Java Class ALCSPrinterMessage.....	43

Introduction

Traditionally WebSphere applications that needed to connect to ALCS based applications had to utilize specialized communication adapters which implemented legacy airline protocols, such as MATIP, or they had to use MQ Series connectivity. In general this type of setup works quite well for smaller installations but has limited scalability and throughput because the highly interactive message exchange between WebSphere and ALCS is carried out over a network connection. Some optimization is achievable by locating the WebSphere Application Server instance on a Linux IFL on the same zSeries machine as ALCS and taking advantage of the fact that in this configuration the Open Systems Adapter (OSA) is routing the messages internally instead of sending them through the physical network.

The introduction of the WebSphere Optimized Local Adapter (OLA) in WebSphere Application Server for z/OS version 7.0.0.4 provides a highly optimized alternative to the previous configuration. Originally the OLA was designed for CICS applications but it can also be used for ALCS in the same manner. To take advantage of the OLA the WebSphere Application Server for z/OS instance is deployed in the same z/OS image as ALCS which allows the OLA to use cross-memory services to exchange messages between the two application environments. This message exchange takes place at memory speed and will therefore provide very good performance. Furthermore the co-location of WebSphere Application Server for z/OS and ALCS in the same z/OS image will provide enhanced availability and reliability and be less complex than a distributed setup.

The following diagram shows a high level view of the environment.



This document is not meant to replace the WebSphere Application Server for z/OS and ALCS product documentation but to supplement them and explain how the two products work together and to give some initial ideas on how to utilize the OLA. For additional details, especially concerning the configuration of the ALCS WAS Bridge, please consult the respective product documentation.

Message Flow

Inbound Message Flow (WAS -> ALCS)

Inbound messages from WebSphere to ALCS are routed through the OLA adapter using the JCA Common Client Interface (CCI) APIs. The destination ALCS WAS Bridge is specified by setting the properties of the `ConnectionSpecImp` and `InteractionSpecImp` objects and need to match the configuration record of the ALCS WAS Bridge.

The request message is sent to ALCS through the OLA `execute()` method which upon success will return the ALCS response as return parameter. However, it is important to note that under certain circumstances ALCS might not be able to return its response right away and will instead only send an acknowledgement back. This situation can arise with long running ALCS applications that are unable to return a response within a pre-configured timeout value. A timeout value was implemented to prevent the OLA `execute()` method from blocking indefinitely, which would block a WebSphere servant thread as well as an ALCS TCB. In this instance the ALCS response is sent through a separate outbound connection.

It is also possible that an ALCS application will respond with several separate messages, in which case only the first message will be returned by the `execute()` method and the remaining ones are sent by ALCS through a separate outbound connection.

This behavior requires some special consideration when implementing the WebSphere application because it will have to be able to retrieve the asynchronously sent response which is delivered through a separate WebSphere servant thread. The easiest approach is to implement a simple queuing mechanism in order to pass the response from the receive thread to the original request thread. It is quite easy to contain these details in a utility layer and an example is given in the sample section of this document.

The basic steps to send a message are as follows:

1. Retrieve a `ConnectionFactory` object for the resource adapter
2. Instantiate a `ConnectionSpecification` object and initialize it
3. Instantiate an `InteractionSpecification` object and initialize it
4. Get a `Connection` object from the `ConnectionFactory`
5. Create an `Interaction`
6. Instantiate a `Record` object and initialize it with the data to be sent
7. Invoke the `execute()` method to send the request and to receive the response
8. Close the `Interaction`
9. Close the `Connection`

The following is the abbreviated code that implements the steps outlined above.

```
InitialContext ctx = new InitialContext();
ConnectionFactory cf = (ConnectionFactory)ctx.lookup("eis/ola");

ConnectionSpecImp csi = new ConnectionSpecImpl();
csi.setRegisterName("ALCSWASPIPE2");
csi.setConnectionWaitTimeout(10);
```

```

InteractionSpecImp isi = new InteractionSpecImpl();
isi.setServiceName("alcsz002");

Connection c = cf.getConnection(csi);
Interaction i = c.createInteraction();

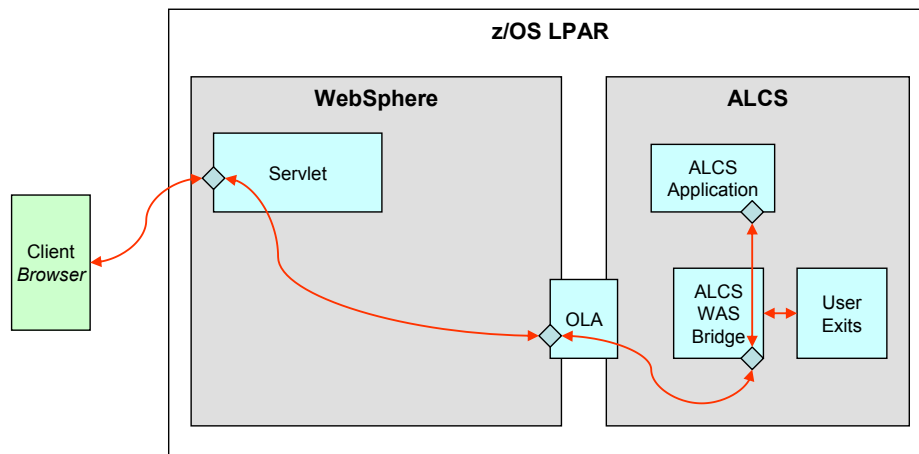
IndexedRecordImpl iri = new IndexedRecordImpl();
iri.add(request);

response = i.execute(isi, iri);

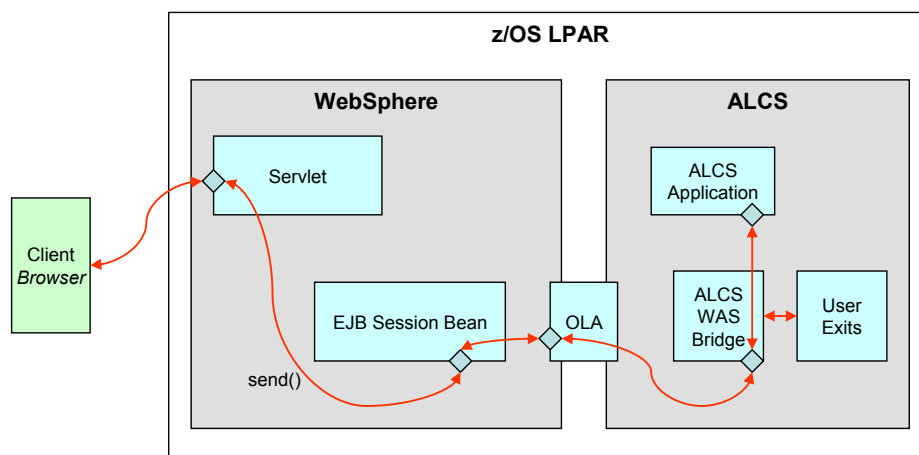
i.close();
c.close();

```

Looking at a Servlet example there are two ways the OLA can be invoked to send a message to ALCS. In the first example below the Servlet directly invokes the OLA through the JCA Common Client Interface (CCI) calls outlined above.



In the second example below the OLA interaction and JCA CCI calls are contained in a Stateless Session Bean and the Servlet invokes the OLA indirectly by obtaining the Stateless Session Bean instead.

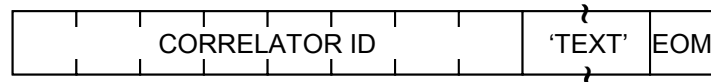


Generally either approach is fine but the second implementation has the advantage of isolating the OLA interaction details in a Stateless Session Bean and provide a simpler to use *send()*

method to the Servlet. In case of changes in the interaction with the OLA they are limited to one location and do not necessarily affect multiple Servlets, or other client applications. This approach also provides a way to implement the exception processing where the ALCS response is being sent asynchronously and it can hide this complexity from the WebSphere applications.

Request Message Format

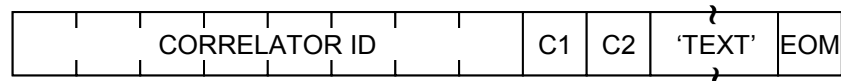
Messages sent to ALCS through the OLA have to adhere to the following message format:



Each message is prefixed with an 8 byte correlator header, followed by the actual message, and then completed with the appropriate End of Message (EOM) character. The format and codepage of the message has to be agreed upon between the client application and the ALCS application. For example the message could be a simple ALCS application command, an EDIFACT formatted message, or something else. For an explanation of the correlator please see the section "Correlator" below.

Response Message Format

Responses sent to WebSphere through the OLA adhere to the following message format:



Each message is prefixed with an 8 byte correlator header, followed by the traditional two byte message prefix which indicates how the message is to be displayed, followed by the actual message, and then completed with the appropriate End of Message (EOM) character. The format and codepage of the message has to be agreed upon between the client application and the ALCS application. For example the message could be a simple ALCS application response, an EDIFACT formatted message, or something else. For an explanation of the correlator please see the section "Correlator" below.

In the event ALCS is unable to return the actual application response a pre-defined one by acknowledgement is returned instead. The acknowledgement has the following format (in EBCDIC).



Outbound Message Flow (ALCS -> WAS)

When ALCS sends an application message to WebSphere through the OLA, the OLA will instantiate an EJB Stateless Session bean through a JNDI lookup and then invoke its *execute()* method. The JNDI name of the Stateless Session Bean is configured in the configuration record

ALCS User Exits

The ALCS WAS Bridge installation wide monitor exits USRWAS3, USRWAS4, USRWAS5, and USRWAS6 allow the customization of the behavior of the WAS Bridge.

User Exit	Description
USRWAS3	WAS input bridge address exit.
USRWAS4	WAS input bridge format exit.
USRWAS5	WAS output bridge address exit.
USRWAS6	WAS output bridge format exit.

User exit USRWAS3 is called upon receiving a message from the OLA and is meant to perform the assignment of a CRI to route the received message. IBM recommends using all, or part, of the 8 byte correlator present in the input message to perform the CRI assignment. For example 3 bytes of the correlator could represent the CRI. A sample user exit with this behavior is provided in module DXCUWAS3.

User exit USRWAS5 is the opposite of user exit USRWAS3 and is called when a message is to be sent to the OLA. The user exit should generate the appropriate 8 byte correlator and IBM recommends using the CRI to create all, or part, of the 8 byte correlator. For example 3 bytes of the 8 byte correlator could represent the CRI. Other information such as terminal type could be represented in the correlator as well. A sample user exist with this behavior is provided in module DXCUWAS5.

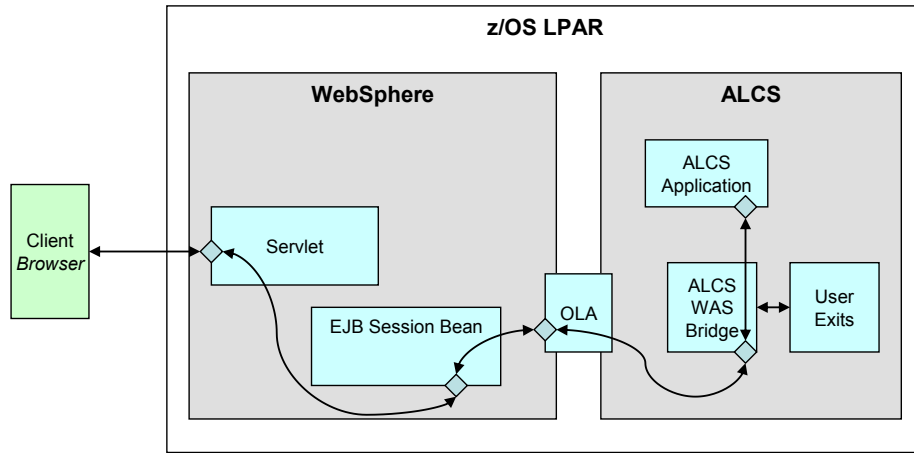
User exits USRWAS4 and USRWAS6 can be used to reformat input and output messages, respectively. Such reformatting could entail the translation between ASCII and EBCDIC, for example. Sample user exists are provided in modules DXCUWAS4 and DXCUWAS6.

Usage Scenarios

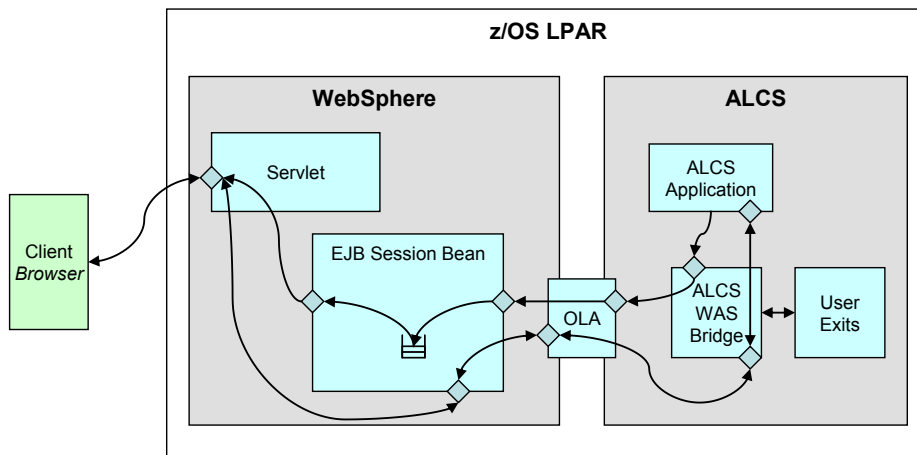
Web Services. Servlets, JSPs

The previous sections of this document used a Servlet as an example of a WebSphere application that could interact with an ALCS application. Web Services and JSPs share a similar application pattern and the same basic approach is suitable for both types of applications.

The diagram below shows the basic setup and message flow used for this implementation approach again.

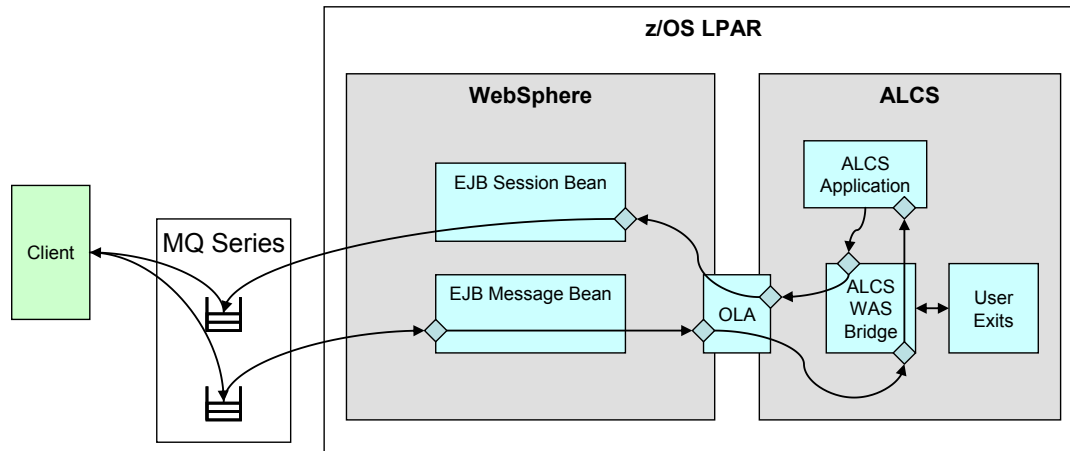


For the scenario to be fool-proof it needs to be able to handle the circumstance where ALCS is returning the response through a separate outbound connection. The more complete scenario below implements this outbound channel and provides the WebSphere application with a simple API to retrieve asynchronous responses, if needed.



MQ Series, JMS

Another usage scenario is based on a messaging approach where the client interaction with WebSphere and ALCS is done through a set of MQ Series queues, as shown in the diagram below.



In this scenario the Stateless Session Bean which receives messages from ALCS is forwarding them to an external MQ Series queue from which the client can retrieve them. The Stateless Session Bean could perform a number of transformation tasks on the message, prior to forwarding it to the MQ Series queue. For example an EDIFACT formatted ALCS message could be transformed into an XML message before being placed on a MQ Series queue.

In the opposite direction an EJB Message Bean is tied to an MQ Series queue and is automatically invoked whenever a new message is placed on the MQ Series queue. The Message Bean will then invoke the OLA to send the message to ALCS. Here again the Message Bean could perform a number of transformation tasks on the message prior to forwarding it to ALCS. For example an XML formatted message could be transformed into an EDIFACT formatted ALCS message prior to sending it to ALCS.

EJB3 Considerations

The EJB3 specification has simplified the coding of J2EE applications tremendously and most likely new applications will no longer be coded based on the prior EJB 2.1 specification. The OLA is compatible with both, EJB 2.1 as well as EJB 3 applications, but EJB 3 applications require some special considerations to work properly.

In the current release the OLA invokes a Stateless Session Bean based on the EJB 2.1 specification and therefore an EJB3 Stateless Session Bean needs to be coded to expose the EJB 2.1 compatible interfaces.

The simplest approach to do so is to use the `@RemoteHome` annotation shown below.

```
@RemoteHome(ExecuteHome.class)
@Stateless
public class ALCSConnectionBean implements ALCSConnection {
    . . .
}
```

If the same Stateless Session Bean is used for inbound and outbound communication then there is an additional factor to consider. During tests it was discovered that there is a problem when a Stateless Session Bean exposes its remote interface as EJB 3 and EJB 2.1 interfaces and the OLA will be unable to invoke the Stateless Session Bean, even though the `@RemoteHome` annotation is used. To resolve the problem the Stateless Session Bean interface used by the WebSphere application has to be defined as Local. This way the OLA side of the Stateless Session Bean is using a remote EJB 2.1 interface and the WebSphere application side of the Stateless Session Bean is using a local EJB 3 interface. This approach has worked quite well and is used in the sample application listed below.

The following code is used for the Stateless Session Bean declaration to achieve this.

```
@RemoteHome(ExecuteHome.class)
@Local(ALCSConnection.class)
@Stateless
public class ALCSConnectionBean implements ALCSConnection {
    . . .
}
```

Transaction Management

The usage of the OLA with ALCS does not support JCA transaction management and it is necessary to indicate this for every Stateless Session Bean that invokes the OLA.

For EJB 3 based Stateless Session Beans this can be done through an annotation as follows.

```
@TransactionAttribute(TransactionAttributeType.NOT_SUPPORTED)
public class ALCSConnectionBean implements ALCSConnection {
    . . .
}
```

Instead of annotations the transaction support can also be specified in the EJB Deployment Descriptor as follows.

```
<ejb-jar version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/ejb-jar_3_0.xsd">
    <display-name>ALCSConnection</display-name>
    <assembly-descriptor>
        <container-transaction>
            <method>
```

```
        <ejb-name>ALCSConnectionBean</ejb-name>  
        <method-name>*</method-name>  
    </method>  
    <trans-attribute>NotSupported</trans-attribute>  
</container-transaction>  
</assembly-descriptor>  
</ejb-jar>
```

Prerequisites

ALCS

The necessary Optimized Local Adapter support is contained in the *WAS Bridge* module which is part of APAR PK83249 and requires ALCS version 2.4.1. In addition APAR PK86999 contains the sample skeleton user exits for the *WAS Bridge*.

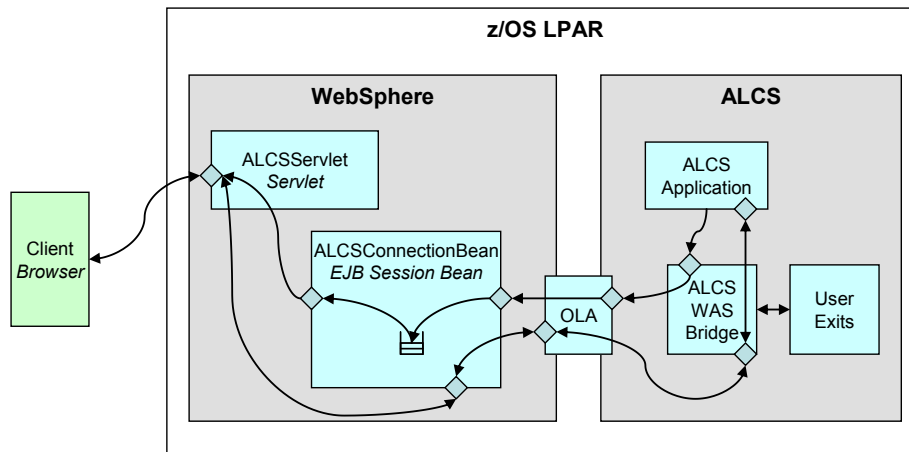
WebSphere

The Optimized Local Adapter is part of APAR PK85842 which is included in Service Level 7.0.0.4 of WebSphere Application Server for z/OS.

Sample Servlet Application

Overview

The following diagram shows the design of a sample Servlet application which provides a browser based user interface to send a request to ALCS and to display the corresponding response. The corresponding source code is included below.

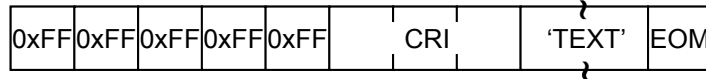


The ALCSServlet is the heart of the application and gets invoked by a user connecting to its URL through a browser. Once the user submits the Web form the ALCSServlet invokes the *send()* method of the ALCSConnectionBean EJB to send the submitted command to ALCS through a call to the OLA and in return it receives the response as return value.

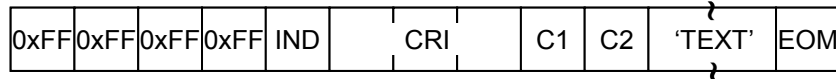
If the return value from the *send()* call is *null* it means ALCS was not able to return the response within the pre-set timeout. In this case the response from ALCS is sent through the OLA and received by the ALCSConnectionBean *execute()* method. The *execute()* method places the received message into the receive queue associated with the CRI which originated the message. The Servlet is attempting to receive the response asynchronously through a call to the *receive()* method of the ALCSConnectionBean EJB. The *receive()* method of the ALCSConnectionBean EJB will retrieve the message from the receive queue, with an optional timeout to wait for it to become available.

Message Format and Correlator

For inbound messages to ALCS the correlator starts with five bytes with a fixed value of 0xFF. It is then followed by the three byte hex value of the CRI that is the destination for this message. The CRI value is not translated between EBCDIC and ASCII. This concludes the 8 byte correlator and the actual payload message follows, which is then terminated with an End of Message (EOM) character. Depending on the type of terminal and ALCS application the EOM character could have different values but for this sample application it is fixed on the value 0x4E (EOM Complete).



For outbound messages to WebSphere the correlator starts with four bytes with a fixed value of 0xFF. It is followed by a one byte value which indicates the type of the terminal (0x01 = ALC, 0x03 = 3270). Following that is the three byte hex value of the CRI that is the origin of this message. The CRI value is not translated between EBCDIC and ASCII. This concludes the 8 byte correlator. The following two bytes represent the standard C1/C2 control characters that indicate the output behavior of the message. The actual payload message follows and is finally terminated with one of the End of Message characters.



Code Listings

ALCS User Exit DXCUWAS3

```

* DAN WEBER APPLICATION                                *DANWEBER 00001001
USRWAS3 TITLE ' *** USRWAS3 INSTALLATION-WIDE MONITOR EXIT *** ' 00002000
*=====* 00003000
*     ALCS INSTALLATION-WIDE MONITOR EXIT USRWAS3                * 00004000
*=====* 00005000
*     APAR AK86999  ADD WASBRIDGE SUPPORT                        @AK86999 00006000
*-----* 00007000
*                                     * 00008000
*     THE ALCS WASBRIDGE CALLS THIS EXIT WHEN ALCS              * 00009000
*     RECEIVES A MESSAGE FROM WAS.                               * 00010000
*                                     * 00020000
*-----* 00030000
*     EJECT ,                                                    00040000
*-----* 00050000
*                                     * 00060000
*     THIS EXIT MUST NOT MODIFY REGISTERS R11 THROUGH R12.      * 00070000
*     THIS EXIT MUST RESTORE ALL REGISTERS EXCEPT R15.        * 00080000
*                                     * 00090000
*     ENTRY CONDITIONS:                                         * 00100000
*     -----                                                  * 00110000
*                                     * 00120000
*     R01 - PARAMETER LIST ADDRESS                               * 00130000
*     PARM1 : ENTRY CONTROL BLOCK (ECB) ADDRESS (EB0EB)         * 00140000
*     PARM2 : ADDRESS OF THE COMMUNICATION TABLE ENTRY         * 00150000
*             FOR THE ORIGINATING WAS RESOURCE                   * 00160000
*     PARM3 : NULL                                              * 00170000
*     PARM4 : ADDRESS OF THE INPUT MESSAGE BLOCK                 * 00180000
*             ATTACHED TO THE ECB OR ADDRESS OF THE INPUT       * 00190000
*             MESSAGE HEAP STORAGE AREA                          * 00200000
*                                     * 00210000
*     R12 - INSTALLATION-WIDE MONITOR EXIT INFORMATION          * 00220000
*             TABLE ADDRESS (IW0IT)                             * 00230000
*                                     * 00240000
*     R13 - CALLING PROGRAM SAVE AREA ADDRESS                    * 00250000
*                                     * 00260000
    
```

```

*      R15 - ADDRESS OF THIS MODULE                * 00270000
*                                                    * 00280000
*-----* 00290000
*      EJECT ,                                     * 00300000
*-----* 00310000
*                                                    * 00320000
*      EXIT CONDITIONS:                           * 00330000
*      -----* 00340000
*                                                    * 00350000
*      ALL REGISTERS SAME AS ON ENTRY EXCEPT R15: * 00360000
*                                                    * 00370000
*      R15 = 0 USE THE ADDRESS OF THE WAS RESOURCE * 00380000
*                                                    * 00390000
*      R15 = 4 USE THE RETURNED TERMINAL ADDRESS   * 00400000
*                                                    * 00410000
*      R15 = 8 PROCESSING IS COMPLETE, DISCARD THE MESSAGE * 00420000
*                                                    * 00430000
*      YOU MUST SET THE FOLLOWING RETURN PARAMETERS WHEN R15 = 4: * 00440000
*                                                    * 00450000
*      PARM5 : THE ADDRESS OF A 4-BYTE FIELD CONTAINING * 00460000
*              THE CRI OF THE ORIGINATING TERMINAL   * 00470000
*              IN THE LOW-ORDER 3 BYTES              * 00480000
*                                                    * 00490000
*-----* 00500000
*      EJECT ,                                     * 00510000
*=====* 00520000
*      EJECT ,                                     * 00530000
*      DXCUHDR NAME=USRWAS3,ENTRY=UWAS000         * 00540000
*      EJECT ,                                     * 00550000
*=====* 00560000
*                                                    * 00570000
*      USER HEADER COMMENTS SECTION STARTS HERE   * 00580000
*                                                    * 00590000
*-----* 00600000
*                                                    * 00610000
*                                                    * 00620000
*                                                    * 00630000
*-----* 00640000
*                                                    * 00650000
*      USER HEADER COMMENTS SECTION ENDS HERE     * 00660000
*                                                    * 00670000
*=====* 00680000
*      EJECT ,                                     * 00690000
*=====* 00700000
*                                                    * 00710000
*      ALCS MACRO DEFINITIONS                      * 00720000
*                                                    * 00730000
*=====* 00740000
*      SPACE 1                                     * 00750000
*      MNØSV REG=R13          MONITOR SAVE AREA DSECT * 00760000
*      PRINT NOGEN                                                  * 00770000
*      SPACE 1                                     * 00780000
*      MNØSV DSECT ,          CONTINUE WORK AREA DSECT * 00790000
*      SPACE 1                                     * 00800000
*      PARMSAVE DC F'Ø'          INPUT PARAMETER LIST ADDRESS * 00810000
*      PARMLIST DC 8F'Ø'        CALLABLE SERVICES PARAMETER LIST * 00820000
*      SPACE 1                                     * 00830000
*                                                    * 00840000
*      USER SAVE AREA EXTENSIONS GO HERE

```

```

*
*
*          END OF USER SAVE AREA EXTENSIONS          00850000
*
*          SPACE 1                                     00860000
*          DC    0D'0'                                00870000
*          SPACE 1                                     00880000
*          SPACE 1                                     00890000
USRSVAV EQU *-MN0SV                                  00900000
USRWAS3 CSECT ,                                       RESTORE CSECT    00910000
*          SPACE 1                                     00920000
PARMS   DSECT ,                                       DSECT FOR PARAMETER LIST 00930000
PARM1   DS    F                                       PARAMETER 1      00940000
PARM2   DS    F                                       PARAMETER 2      00950000
PARM3   DS    F                                       PARAMETER 3      00960000
PARM4   DS    F                                       PARAMETER 4      00970000
PARM5   DS    F                                       PARAMETER 5      00980000
PARM6   DS    F                                       PARAMETER 6      00990000
PARM7   DS    F                                       PARAMETER 7      01000000
PARM8   DS    F                                       PARAMETER 8      01010000
*          SPACE 1                                     *DANWEBER 01011001
WORKCRI DS    F                                       WORK CRI         *DANWEBER 01012001
*          USING PARM5,R01
USRWAS3 CSECT ,                                       RESTORE CSECT    01030000
*          SPACE 1                                     01040000
*=====* 01050000
*
*          END OF ALCS MACRO DEFINITIONS              * 01060000
*
*          END OF ALCS MACRO DEFINITIONS              * 01070000
*
*          END OF ALCS MACRO DEFINITIONS              * 01080000
*=====* 01090000
*          EJECT ,                                     01100000
*=====* 01110000
*
*          USER MACRO DEFINITIONS SHOULD BE INSERTED BELOW * 01120000
*
*          USER MACRO DEFINITIONS SHOULD BE INSERTED BELOW * 01130000
*
*          USER MACRO DEFINITIONS SHOULD BE INSERTED BELOW * 01140000
*=====* 01150000
*          SPACE 1                                     01160000
*          SPACE 1                                     01170000
*=====* 01180000
*
*          END OF USER MACRO DEFINITIONS              * 01190000
*
*          END OF USER MACRO DEFINITIONS              * 01200000
*
*          END OF USER MACRO DEFINITIONS              * 01210000
*=====* 01220000
*          PRINT NOGEN                                01230000
USRWAS3 CSECT ,                                       RESTORE CSECT    01240000
*          EJECT ,                                     01250000
*=====* 01260000
*
*          USER OBJECT COMMENTS SHOULD BE INSERTED BELOW * 01270000
*
*          USER OBJECT COMMENTS SHOULD BE INSERTED BELOW * 01280000
*
*          USER OBJECT COMMENTS SHOULD BE INSERTED BELOW * 01290000
*=====* 01300000
*          SPACE 1                                     01310000
*          SPACE 1                                     01320000
*=====* 01330000
*
*          END OF USER OBJECT COMMENTS                * 01340000
*
*          END OF USER OBJECT COMMENTS                * 01350000
*
*          END OF USER OBJECT COMMENTS                * 01360000
*=====* 01370000
*          EJECT ,                                     01380000
UWAS000 DC    0H'0'                                    01390000
*          DXCSAVE SAVE,PUSH,                         SAVE CALLERS REGISTERS -01400000

```

```

        ID=USER,          AND CHAIN THE USER SAVE AREA      -01410000
        SIZE=512,        LENGTH DEFINED IN SAVE AREA      -01420000
        WORKREG=R02                                           01430000
SPACE 1
DROP R15              DROP PREVIOUS BASE REGISTER          01440000
USING USRWAS3,R10    01450000
LR R10,R15           USE R10 AS BASE REGISTER              01460000
SR R15,R15           SET GOOD RETURN CODE                  01470000
ST R01,PARMSAVE     SAVE INPUT PARAMETER LIST              01480000
SPACE 1              01490000
=====* 01500000
* 01510000
* 01520000
* START OF USER CODE 01530000
* 01540000
=====* 01550000
SPACE 1              01560000
L R14,PARM4          LOAD ADDRESS OF MESSAGE *DANWEBER 01570000
CLC =X'FFFFFFF',X'15'(R14) IS THERE A CORRELATOR *DANWEBER 01580000
BNE UWAS9000         BRANCH IF NOT *DANWEBER 01590000
SPACE 1              *DANWEBER 01591000
MVC WORKCRI+1(3),5+X'15'(R14) COPY CRI FROM CORR. *DANWEBER 01592000
MVI WORKCRI,X'00'    SET TOP BYTE TO ZEROES *DANWEBER 01593000
SPACE 1              *DANWEBER 01594000
LA R14,WORKCRI       LOAD ADDRESS OF CRI *DANWEBER 01595000
ST R14,PARM5         SAVE IN PARM5 *DANWEBER 01596000
SPACE 1              *DANWEBER 01597000
LA R15,4             MESSAGE CONTAINS CORRELATOR *DANWEBER 01598000
B UWAS9000           CONTINUE *DANWEBER 01599000
SPACE 1              01600000
=====* 01610000
* 01620000
* END OF USER CODE 01630000
* 01640000
=====* 01650000
SPACE 1              01660000
UWAS9000 DC 0H'0'     01670000
DXCSAVE POP,RESTORE, RESTORE CALLERS REGISTERS -01680000
          EXCEPT=((R15)) (EXCEPT RETURN INFORMATION) 01690000
BR R14              RETURN TO ALCS MONITOR 01700000
EJECT ,             01710000
=====* 01720000
* 01730000
* START OF USER CONSTANTS 01740000
* 01750000
=====* 01760000
SPACE 1              01770000
SPACE 1              01780000
=====* 01790000
* 01800000
* END OF USER CONSTANTS 01810000
* 01820000
=====* 01830000
EJECT ,             01840000
LTORG ,             01850000
DXCUEND ,           01860000
END                 01870000
    
```

ALCS User Exit DXCUWAS5

```

* DAN WEBER APPLICATION *DANWEBER 00001000
USRWAS5 TITLE ' *** USRWAS5 INSTALLATION-WIDE MONITOR EXIT *** ' 00002000
*=====* 00003000
* ALCS INSTALLATION-WIDE MONITOR EXIT USRWAS5 * 00004000
*=====* 00005000
* APAR AK86999 ADD WASBRIDGE SUPPORT @AK86999 00006000
*-----* 00007000
* * 00008000
* THE ALCS WASBRIDGE CALLS THIS EXIT WHEN ALCS * 00009000
* SENDS A MESSAGE TO WAS. * 00010000
* * 00020000
*-----* 00030000
* EJECT , 00040000
*-----* 00050000
* * 00060000
* THIS EXIT MUST NOT MODIFY REGISTERS R11 THROUGH R12. * 00070000
* THIS EXIT MUST RESTORE ALL REGISTERS EXCEPT R15. * 00080000
* * 00090000
* ENTRY CONDITIONS: * 00100000
* ----- * 00110000
* * 00120000
* R01 - PARAMETER LIST ADDRESS * 00130000
* PARM1 : ENTRY CONTROL BLOCK (ECB) ADDRESS (EB0EB) * 00140000
* PARM2 : ADDRESS OF THE COMMUNICATION TABLE ENTRY * 00150000
* FOR THE DESTINATION WAS RESOURCE * 00160000
* PARM3 : ADDRESS OF THE COMMUNICATION TABLE ENTRY * 00170000
* FOR THE DESTINATION TERMINAL RESOURCE, * 00180000
* IF ANY. OTHERWISE NULL * 00190000
* PARM4 : ADDRESS OF THE OUTPUT MESSAGE BLOCK * 00200000
* ATTACHED TO THE ECB OR ADDRESS OF THE OUTPUT * 00210000
* MESSAGE HEAP STORAGE AREA * 00220000
* * 00230000
* R12 - INSTALLATION-WIDE MONITOR EXIT INFORMATION * 00240000
* TABLE ADDRESS (IW0IT) * 00250000
* * 00260000
* R13 - CALLING PROGRAM SAVE AREA ADDRESS * 00270000
* * 00280000
* R15 - ADDRESS OF THIS MODULE * 00290000
* * 00300000
*-----* 00310000
* EJECT , 00320000
*-----* 00330000
* * 00340000
* EXIT CONDITIONS: * 00350000
* ----- * 00360000
* * 00370000
* ALL REGISTERS SAME AS ON ENTRY EXCEPT R15: * 00380000
* * 00390000
* R15 = 0 DO NOT ADD ANY HEADER * 00400000
* * 00410000
* R15 = 4 ADD THE SUPPLIED 8-BYTES HEADER * 00420000
* * 00430000
* R15 = 8 PROCESSING IS COMPLETE, DISCARD THE MESSAGE * 00440000
* * 00450000
* YOU MUST SET THE FOLLOWING RETURN PARAMETERS WHEN R15 = 4: * 00460000
* * 00470000

```

```

*          PARM5 : THE ADDRESS OF THE 8-BYTES HEADER          * 00480000
*                                                             * 00490000
*-----* 00500000
          EJECT ,                                           00510000
*=====* 00520000
          EJECT ,                                           00530000
          DXCUHDR NAME=USRWAS5,ENTRY=UWAS000                00540000
          EJECT ,                                           00550000
*=====* 00560000
*                                                             * 00570000
          USER HEADER COMMENTS SECTION STARTS HERE          * 00580000
*                                                             * 00590000
*-----* 00600000
*                                                             * 00610000
*                                                             * 00620000
*                                                             * 00630000
*-----* 00640000
*                                                             * 00650000
          USER HEADER COMMENTS SECTION ENDS HERE            * 00660000
*                                                             * 00670000
*=====* 00680000
          EJECT ,                                           00690000
*=====* 00700000
*                                                             * 00710000
          ALCS MACRO DEFINITIONS                             * 00720000
*                                                             * 00730000
*=====* 00740000
          SPACE 1                                           00750000
          MN0SV REG=R13          MONITOR SAVE AREA DSECT    00760000
          PRINT NOGEN                                           00770000
          SPACE 1                                           00780000
MN0SV    DSECT ,          CONTINUE WORK AREA DSECT          00790000
          SPACE 1                                           00800000
PARMSAVE DC   F'0'          INPUT PARAMETER LIST ADDRESS    00810000
PARMLIST DC   8F'0'        CALLABLE SERVICES PARAMETER LIST 00820000
          SPACE 1                                           00830000
*                                                             * 00840000
*          USER SAVE AREA EXTENSIONS GO HERE                00850000
*
*          END OF USER SAVE AREA EXTENSIONS                  00860000
          SPACE 1                                           00870000
          DC   0D'0'          FORCE LENGTH TO DOUBLEWORDS    00880000
          SPACE 1                                           00890000
USRSVAV EQU *-MN0SV                                           00900000
USRWAS5 CSECT ,          RESTORE CSECT                       00910000
          SPACE 1                                           00920000
PARMS   DSECT ,          DSECT FOR PARAMETER LIST           00930000
PARM1   DS   F          PARAMETER 1                         00940000
PARM2   DS   F          PARAMETER 2                         00950000
PARM3   DS   F          PARAMETER 3                         00960000
PARM4   DS   F          PARAMETER 4                         00970000
PARM5   DS   F          PARAMETER 5                         00980000
PARM6   DS   F          PARAMETER 6                         00990000
PARM7   DS   F          PARAMETER 7                         01000000
PARM8   DS   F          PARAMETER 8                         01010000
          SPACE 1
WORKCOR DC   XL8'00'          WORK CORRELATOR                *DANWEBER 01011000
          USING PARMS,R01                                         *DANWEBER 01012000
          USING PARMS,R01                                         01020000
USRWAS5 CSECT ,          RESTORE CSECT                       01030000
    
```

```

SPACE 1 01040000
*=====* 01050000
* 01060000
* END OF ALCS MACRO DEFINITIONS * 01070000
* 01080000
*=====* 01090000
EJECT , 01100000
*=====* 01110000
* 01120000
* USER MACRO DEFINITIONS SHOULD BE INSERTED BELOW * 01130000
* 01140000
*=====* 01150000
SPACE 1 01160000
SPACE 1 01170000
*=====* 01180000
* 01190000
* END OF USER MACRO DEFINITIONS * 01200000
* 01210000
*=====* 01220000
PRINT NOGEN 01230000
USRWAS5 CSECT , RESTORE CSECT 01240000
EJECT , 01250000
*=====* 01260000
* 01270000
* USER OBJECT COMMENTS SHOULD BE INSERTED BELOW * 01280000
* 01290000
*=====* 01300000
SPACE 1 01310000
SPACE 1 01320000
*=====* 01330000
* 01340000
* END OF USER OBJECT COMMENTS * 01350000
* 01360000
*=====* 01370000
EJECT , 01380000
UWAS000 DC 0H'0' 01390000
DXCSAVE SAVE,PUSH, SAVE CALLERS REGISTERS -01400000
ID=USER, AND CHAIN THE USER SAVE AREA -01410000
SIZE=512, LENGTH DEFINED IN SAVE AREA -01420000
WORKREG=R02 01430000
SPACE 1 01440000
DROP R15 DROP PREVIOUS BASE REGISTER 01450000
USING USRWAS5,R10 01460000
LR R10,R15 USE R10 AS BASE REGISTER 01470000
SR R15,R15 SET GOOD RETURN CODE 01480000
ST R01,PARMSAVE SAVE INPUT PARAMETER LIST 01490000
SPACE 1 01500000
*=====* 01510000
* 01520000
* START OF USER CODE * 01530000
* 01540000
*=====* 01550000
SPACE 1 01560000
SPACE 1 *NEW***** 01560000
***** BUILD 8-BYTE CORRELATOR *NEW***** 01560000
* BYTE 1-4 FFFFFFFF *NEW***** 01560000
* BYTE 5 DEVICE TYPE *NEW***** 01560000
* 01 = ALC DISPLAY *NEW***** 01560000

```

ALCS and WebSphere OLA



```

*           02 = ALC PRINTER                *NEW***** 01560000
*           03 = 3270 DISPLAY                *NEW***** 01560000
*           04 = 3270 PRINTER                *NEW***** 01560000
*
*   BYTE 6-8   CRI                *NEW***** 01560000
*   SPACE 1                *NEW***** 01560000
*   L   R14,PARM3          LOAD ADDRESS OF TERMINAL *DANWEBER-01570001
*                               COMMS ENTRY - IF ANY *DANWEBER 01580000
*   LTR R14,R14           DOES THIS EXIST          *DANWEBER 01590000
*   BZ  UWAS9000          BRANCH IF NOT - FINISHED *DANWEBER 01591000
*   COØRE REG=R14                *DANWEBER 01592000
*   TRMEQ ,                DEVICE TYPE EQUATES     *NEW***** 01592102
*   MVC  WORKCOR+5(3),RECOCRI+1 COPY CRI          *DANWEBER 01593000
*   MVI  WORKCOR+4,X'01'    ALC DEVICE TYPE        *NEW***** 01594000
*   TM   REC1ALC,L'REC1ALC  IS IT ALC DEVICE TYPE  *NEW***** 01595000
*   BO   UWAS100           YES - DEVICE TYPE IS OKAY *NEW***** 01598000
*   SPACE 1                *DANWEBER 01599000
*   MVI  WORKCOR+4,X'03'    3270 DEVICE TYPE      *DANWEBER 01599100
*   SPACE 1                *DANWEBER 01599200
UWAS100 DC  0H'0'                *DANWEBER 01599300
*   TM   REC1TPP,L'REC1TPP  PRINTER                *DANWEBER 01599400
*   BZ  UWAS200           BRANCH IF NOT            *DANWEBER 01599500
*   SPACE 1                *DANWEBER 01599600
*   SR   R00,R00           SET TO ZEROES           *DANWEBER 01599700
*   IC   R00,WORKCOR+4     LOAD INDICATOR          *DANWEBER 01599800
*   AH   R00,=Y(1)        INCREMENT WITH ONE.... *DANWEBER 01599900
*   STC  R00,WORKCOR+4     STORE UPDATED INDICATOR *DANWEBER 01600000
*   SPACE 1                *DANWEBER 01600100
UWAS200 DC  0H'0'                *DANWEBER 01600200
*   MVC  WORKCOR(4),=X'FFFFFFF' INDICATE CORRELATOR *DANWEBER 01600300
*   LA   R14,WORKCOR       LOAD ADDRESS OF WORK CORR. *DANWEBER 01600400
*   ST   R14,PARM5         SAVE IN PARM5           *DANWEBER 01600500
*   SPACE 1                *DANWEBER 01600600
*   LA   R15,4             CORRELATOR NEEDED       *DANWEBER 01600700
*   B    UWAS9000          CONTINUE                *DANWEBER 01600800
*   SPACE 1                01601000
*=====* 01610000
*
*
*   END OF USER CODE                * 01630000
*
*=====* 01640000
*=====* 01650000
*   SPACE 1                01660000
UWAS9000 DC  0H'0'                01670000
*   DXCSAVE POP,RESTORE,    RESTORE CALLERS REGISTERS -01680000
*   EXCEPT=((R15))        (EXCEPT RETURN INFORMATION) 01690000
*   BR   R14                RETURN TO ALCS MONITOR        01700000
*   EJECT ,                01710000
*=====* 01720000
*
*
*   START OF USER CONSTANTS        * 01730000
*
*
*=====* 01740000
*
*=====* 01750000
*
*=====* 01760000
*   SPACE 1                01770000
*   SPACE 1                01780000
*=====* 01790000
*
*
*   END OF USER CONSTANTS        * 01800000
*
*
*=====* 01810000
*
*=====* 01820000
*
*=====* 01830000

```

```
EJECT , 01840000
LTORG , 01850000
DXCUEND , 01860000
END 01870000
```

WebSphere Servlet ALCSServlet

```
package com.ibm.alcs;

import java.io.IOException;
import java.io.PrintWriter;

import javax.ejb.EJB;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/*
 * Licensed Materials - Property of IBM
 * © Copyright IBM Corp. 2009 - All Rights Reserved.
 * DISCLAIMER:
 * THE FOLLOWING [ENCLOSED] CODE IS SAMPLE CODE CREATED BY IBM
 * CORPORATION. THIS SAMPLE CODE IS NOT PART OF ANY STANDARD IBM PRODUCT
 * AND IS PROVIDED TO YOU SOLELY FOR THE PURPOSE OF ASSISTING YOU IN THE
 * DEVELOPMENT OF YOUR APPLICATIONS. THE CODE IS PROVIDED 'AS IS',
 * WITHOUT WARRANTY OF ANY KIND. IBM SHALL NOT BE LIABLE FOR ANY DAMAGES
 * ARISING OUT OF YOUR USE OF SAMPLE CODE, EVEN IF THEY HAVE BEEN
 * ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
 */

/**
 * Servlet implementation class ALCServlet
 */
public class ALCSServlet extends HttpServlet {

    private static final long serialVersionUID = 1L;

    @EJB
    private ALCSConnection alcs;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public ALCSServlet() {
        super();
    }

    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
        ServletException, IOException {
        String[] cris = null;
    }
}
```

```

System.out.println("ALCSServlet.doGet()");

    cris = alcs.getCRIs();

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    out.println("<html>");
    out.println("<head><title>ALCS Servlet</title></head>");
    out.println("<body onload='document.all[\"command\"].focus();'>");

    out.println("<form method='post' action='ALCSServlet'>");

    out.println("Select CRI <select size='1' name='cri' id='cri'>");
    for(String s : cris) {
        out.printf("<option value='%1$s'>%1$s</option>", s);
    }
    out.println("</select>");
    out.println("<br />");

    out.println("ALCS Command <input type='text' name='command' id='command' /><input
type='submit' value='Submit' />");
    out.println("</form>");

    out.println("</body>");
    out.println("</html>");
}

/**
 * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
 */
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    String[] cris = null;

    String cri = null;
    String command = null;
    String resp = null;
    boolean error = false;
    boolean async = false;

    System.out.println("ALCSServlet.doPost()");

    cris = alcs.getCRIs();

    cri = request.getParameter("cri");
    System.out.println("ALCSServlet.doPost() : cri=" + cri);

    command = request.getParameter("command");
    System.out.println("ALCSServlet.doPost() : command=" + command);

    long t1 = 0;
    long t2 = 0;

    t1 = System.currentTimeMillis();
    try {
        resp = alcs.send(cri, command);
    }

```

```

    } catch(Exception e) {
        error = true;
    }
    t2 = System.currentTimeMillis();

    if((resp == null) && (error == false)) {
        // response might come delayed through async queue
        System.out.println("ALCSServlet.doPost() : need to do async receive");
        resp = alcs.receive(cri, 2000);
        t2 = System.currentTimeMillis();
        async = true;
    }

    System.out.printf("ALCSServlet.doPost() : send/receive: %1$dms\n", t2-t1);

    if(resp == null) {
        System.out.println("ALCSServlet.doPost() : nothing received");
    } else {
        System.out.println("ALCSServlet.doPost() : send/receive done");
    }

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    out.println("<html>");
    out.println("<head><title>ALCS Servlet</title></head>");
    out.println("<body onload='document.all[\"command\"].focus();'>");

    out.println("<form method='post' action='ALCSServlet'>");

    out.println("Select CRI <select size='1' name='cri' id='cri'>");
    for(String s : cris) {
        if(s.equalsIgnoreCase(cri)) {
            out.printf("<option value='%1$s' selected>%1$s</option>", s);
        } else {
            out.printf("<option value='%1$s'>%1$s</option>", s);
        }
    }
    out.println("</select>");
    out.println("<br />");

    out.println("ALCS Command <input type='text' name='command' id='command' /><input
type='submit' value='Submit' />");
    out.println("</form>");

    out.println("<br />");

    if(!error) {
        if(resp != null) {
            out.println("<pre>");
            out.println(resp.substring(2));
            out.println("</pre>");
            out.println("<br />");
            out.printf("<div>send/receive: %1$dms</div>", t2-t1);
            if(async) {
                out.println("<div>*** MESSAGE RECEIVED ASYNCHRONOUSLY ***</div>");
            }
        }
        } else {

```

```

        out.println("<div>*** NO MESSAGE FROM ALCS ***</div>");
    }
} else {
    out.println("<div>*** SEND/RECEIVE ERROR ***</div>");
}

out.println("<br />");
if(resp != null) {
    out.println("<hr />");
    out.println("<br />");

    out.println("<pre>");
    out.println(DataDumper.formatData(resp.getBytes()));
    out.println("</pre>");
}

out.println("</body>");
out.println("</html>");

}
}

```

WebSphere EJB Session Bean Interface ALCSConnection

```

package com.ibm.alcs;

import java.io.IOException;

import javax.ejb.Local;

/*
 * Licensed Materials - Property of IBM
 * © Copyright IBM Corp. 2009 - All Rights Reserved.
 * DISCLAIMER:
 * THE FOLLOWING [ENCLOSED] CODE IS SAMPLE CODE CREATED BY IBM
 * CORPORATION. THIS SAMPLE CODE IS NOT PART OF ANY STANDARD IBM PRODUCT
 * AND IS PROVIDED TO YOU SOLELY FOR THE PURPOSE OF ASSISTING YOU IN THE
 * DEVELOPMENT OF YOUR APPLICATIONS. THE CODE IS PROVIDED 'AS IS',
 * WITHOUT WARRANTY OF ANY KIND. IBM SHALL NOT BE LIABLE FOR ANY DAMAGES
 * ARISING OUT OF YOUR USE OF SAMPLE CODE, EVEN IF THEY HAVE BEEN
 * ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
 */

@Local
public interface ALCSConnection {

    /**
     * Receive a message from the specified CRI. This call will block until a message
     becomes
     * available or the specified timeout expires.
     * @param cri the terminal address from which to receive a message
     * @param timeout the timeout in milliseconds, can be 0 as well. -1 will wait forever.
     * @return the received message, translated to the local codepage. This message
     potentially
    */
}

```

```

    * contains C1/C2 characters, or NULL if the timeout expired.
    */
    public String receive(String cri, long timeout);

    /**
     * Receive a message from the specified CRI. This call will block until a message
    becomes
     * available or the specified timeout expires.
     * @param cri the terminal address from which to receive a message
     * @param timeout the timeout in milliseconds, can be 0 as well. -1 will wait forever.
     * @return the received message, untranslated and including C1/C2 as well as EOM
    characters, or
     * NULL if the timeout expired.
     */
    public byte[] receiveRaw(String cri, long timeout);

    /**
     * Send a message to the ALCS system and returns the response. The message will be
    translated to
     * the host codepage prior to sending.
     * @param cri the terminal address to which to send the message.
     * @param message the message to be sent, formatted in the local codepage and NOT
    containing the
     * EOM (end of message) character.
     * @return the response, translated from the host code page
     * @throws IOException if there was an error sending/receiving
     */
    public String send(String cri, String message) throws IOException;

    /**
     * Send a message to the ALCS system and returns the response. The message will be sent
    without
     * further translation.
     * @param cri the terminal address to which to send the message.
     * @param message the message to be sent, formatted in the codepage suitable for ALCS
    and must
     * also containing the EOM (end of message) character.
     * @return the received message, untranslated and including C1/C2 as well as EOM
    characters
     * @throws IOException if there was an error sending/receiving
     */
    public byte[] sendRaw(String cri, byte[] message) throws IOException;

    /**
     * Clears the receive queue of a specific CRI.
     * @param cri the cri whos queue to clear.
     */
    public void clearReceiveQueue(String cri);

    /**
     * Return a list of all CRIs on this connection
     * @return a list of all CRIs on this connection
     */
    public String[] getCRIs();

```

```
/**
 * Validates a given cri to make sure it's valid for this connection.
 * @param cri the cri to be validated.
 * @return true if the cri is valid, otherwise false.
 */
public boolean isValidCRI(String cri);

/**
 * Entry point for OLA to call with out of bound messages
 * @param message message from OLA
 * @return response to OLA
 */
public byte[] execute(byte[] message);
}
```

WebSphere EJB Session Bean ALCSConnectionBean

```
package com.ibm.alcs;

import java.io.IOException;
import java.util.Hashtable;
import java.util.concurrent.ArrayBlockingQueue;
import java.util.concurrent.TimeUnit;

import javax.annotation.PostConstruct;
import javax.ejb.Local;
import javax.ejb.RemoteHome;
import javax.ejb.Stateless;
import javax.ejb.TransactionAttribute;
import javax.ejb.TransactionAttributeType;
import javax.naming.InitialContext;
import javax.resource.cci.Connection;
import javax.resource.cci.ConnectionFactory;
import javax.resource.cci.Interaction;

import com.ibm.websphere.ola.ConnectionSpecImpl;
import com.ibm.websphere.ola.ExecuteHome;
import com.ibm.websphere.ola.IndexedRecordImpl;
import com.ibm.websphere.ola.InteractionSpecImpl;

/*
 * Licensed Materials - Property of IBM
 * © Copyright IBM Corp. 2009 - All Rights Reserved.
 * DISCLAIMER:
 * THE FOLLOWING [ENCLOSED] CODE IS SAMPLE CODE CREATED BY IBM
 * CORPORATION. THIS SAMPLE CODE IS NOT PART OF ANY STANDARD IBM PRODUCT
 * AND IS PROVIDED TO YOU SOLELY FOR THE PURPOSE OF ASSISTING YOU IN THE
 * DEVELOPMENT OF YOUR APPLICATIONS. THE CODE IS PROVIDED 'AS IS',
 * WITHOUT WARRANTY OF ANY KIND. IBM SHALL NOT BE LIABLE FOR ANY DAMAGES
 * ARISING OUT OF YOUR USE OF SAMPLE CODE, EVEN IF THEY HAVE BEEN
 * ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
 */

/**
```

```
* Session Bean implementation class ALCSConnectionBean2
*/
@RemoteHome(ExecuteHome.class)
@Local(ALCSConnection.class)
@Stateless
@TransactionAttribute(TransactionAttributeType.NOT_SUPPORTED)
public class ALCSConnectionBean implements ALCSConnection {

    private static char[] HEX_CHARS = "0123456789ABCDEF".toCharArray();
    private static byte[] ALCS_ACK = {(byte)0xc1};

    private static String[] CRIS = {"02025A", "02025B"};
    private static String CODEPAGE = "Cp037";
    private static String OLA_JNDI = "eis/ola";
    private static String REGISTER_NAME = "ALCSWASPIPE2";
    private static String SERVICE_NAME = "alcsz002";

    private static Hashtable<String, ArrayBlockingQueue<byte[]>> receiveQueues = null;

    private ConnectionFactory cf = null;
    private ConnectionSpecImpl csi = null;
    private InteractionSpecImpl isi = null;

    /**
     * Default constructor.
     */
    public ALCSConnectionBean() {

        if(receiveQueues == null) {
            System.out.println("ALCSConnectionBean() : Initializing Receive Queues");

            receiveQueues = new Hashtable<String, ArrayBlockingQueue<byte[]>>();

            for(String cri : CRIS) {
                ArrayBlockingQueue<byte[]> queue = new ArrayBlockingQueue<byte[]>(10);
                receiveQueues.put(cri.toUpperCase(), queue);
            }
        }
    }

    @PostConstruct
    public void initialize() {

        // Initialize connector variables which are unique to each instance of this
        SessionBean

        try {
            InitialContext ctx = new InitialContext();

            cf = (ConnectionFactory)ctx.lookup(OLA_JNDI);

            csi = new ConnectionSpecImpl();
            csi.setRegisterName(REGISTER_NAME);
            csi.setConnectionWaitTimeout(5);

            isi = new InteractionSpecImpl();
```

```
        isi.setServiceName(SERVICE_NAME);
    } catch(Exception e) {
        cf = null;
        csi = null;
        isi = null;
        System.out.println("ALCSConnectionBean.initialize() : Error initializing");
        e.printStackTrace();
    }
}

@Override
public void clearReceiveQueue(String cri) {

    ArrayBlockingQueue<byte[]> queue = receiveQueues.get(cri);

    if(queue != null) {
        queue.clear();
    }

}

/**
 * Get an array of strings with all CRIs that can be used on this connection to ALCS
 * @return a list of CRIs that can be used on this connection
 */
@Override
public String[] getCRIs() {

    String[] s = new String[CRIS.length];

    for(int i = 0; i < CRIS.length; i++) {
        s[i] = new String(CRIS[i]);
    }
    return s;
}

@Override
public boolean isValidCRI(String cri) {

    if(cri == null) {
        return false;
    }

    for(String c : CRIS) {
        if(c.equalsIgnoreCase(cri)) {
            return true;
        }
    }

    return false;
}

/**
```

```

* Entrypoint that is executed by OLA for messages originating from ALCS.
* @param message the message received from ALCS.
* @return the response back to ALCS, which should be the single character 'A'.
*/
public byte[] execute(byte[] message) {

    String strCRI = getCRI(message);

    ArrayBlockingQueue<byte[]> queue = receiveQueues.get(strCRI);
    if(queue != null) {
        if(!queue.offer(message)) {
            System.out.println("ALCSConnectionBean.execute() : Receive queue full for
CRI=" + strCRI);
        }
    } else {
        System.out.println("ALCSConnectionBean.execute() : Receives message for unknown
CRI=" + strCRI);
    }

    return ALCS_ACK;

}

/**
* Perform ALC terminal type wrapping at column limit
* @param str the string to wrap
* @param offset the start offset (to exclude C1/C2)
* @param limit the column where to wrap (0 based)
* @return the wrapped string
*/
private String doWrapping(String str, int offset, int limit) {

    StringBuffer sb = new StringBuffer();
    int col = 0;
    int i;
    char chr;

    for(i = 0; ((i < str.length()) && (i < offset)); i++) {
        chr = str.charAt(i);
        sb.append(chr);
    }

    for(i = offset; i < str.length(); i++) {

        chr = str.charAt(i);
        sb.append(chr);

        if(chr == '\n') {
            col = 0;
        }
        else {
            col++;
            if(col > limit) {
                col = 0;
                sb.append('\n');
            }
        }
    }
}

```

```

    }

    return sb.toString();

}

/**
 * Helper method to retrieve CRI in String format. This method is about 20x faster than
 * using String.format()
 * @param data the received data array
 * @return the CRI as String
 */
private String getCRI(byte[] data) {

    if((data == null) || (data.length < 8)) {
        return "";
    }

    char[] chars = new char[6];
    chars[0] = HEX_CHARS[(data[5] & 0xF0) >>> 4];
    chars[1] = HEX_CHARS[data[5] & 0x0F];
    chars[2] = HEX_CHARS[(data[6] & 0xF0) >>> 4];
    chars[3] = HEX_CHARS[data[6] & 0x0F];
    chars[4] = HEX_CHARS[(data[7] & 0xF0) >>> 4];
    chars[5] = HEX_CHARS[data[7] & 0x0F];
    return new String(chars);

}

@Override
public String receive(String cri, long timeout) {

    String strMsg;

    try {
        byte[] msg = receiveQueue(cri, timeout);
        if(msg != null) {

            strMsg = new String(msg, 8, msg.length - 9, CODEPAGE);
            if(msg[4] == 0x01) {
                strMsg = doWrapping(strMsg, 2, 63);
            }

            return strMsg;
        }
    } catch(Exception e) {
        System.out.println("ALCSConnectionBean.receive() : Error during receive");
        e.printStackTrace();
    }

    return null;
}

@Override
public byte[] receiveRaw(String cri, long timeout) {

```

```

    byte[] msg = null;
    byte[] message = receiveQueue(cri, timeout);

    if(message != null) {
        msg = new byte[message.length - 8];
        System.arraycopy(message, 8, msg, 0, message.length - 8);
    }

    return msg;
}

/**
 * Receive message from out of bound queue.
 * @param cri the CRI for which to retrieve a message
 * @param cri the terminal address from which to receive a message
 * @param timeout the timeout in milliseconds, can be 0 as well. -1 will wait forever.
 * @return the received message, as received from ALCS.
 */
private byte[] receiveQueue(String cri, long timeout) {

    byte[] msg = null;

    try {

        ArrayBlockingQueue<byte[]> queue = receiveQueues.get(cri);
        if(queue != null) {

            if(timeout == 0) {
                msg = queue.poll();
            } else if(timeout == -1) {
                msg = queue.take();
            } else {
                msg = queue.poll(timeout, TimeUnit.MILLISECONDS);
            }

        }

    } catch(Exception e) {
        System.out.println("ALCSConnectionBean.receiveQueue() : Error during receive");
        e.printStackTrace();
    }

    return msg;
}

@Override
public String send(String cri, String message) throws IOException {

    byte[] tmp;
    byte[] msg;
    String strMsg = null;

    try {
        tmp = message.getBytes(ALCSSTable.getALCSSTable().getCodepage());
        msg = new byte[tmp.length + 9];
        System.arraycopy(tmp, 0, msg, 8, tmp.length);
        msg[msg.length-1] = 0x4E;
    }

```

```

        msg = sendOLA(cri, msg);

        if(msg != null) {

            strMsg = new String(msg, 8, msg.length - 9,
ALCSSessionTable.getALCSSessionTable().getCodepage());
            if(msg[4] == 0x01) {
                strMsg = doWrapping(strMsg, 2, 63);
            }

            return strMsg;
        }

    } catch(Exception e) {
        System.out.println("ALCSConnectionBean.send() : Error sending/receiving data");
        e.printStackTrace();
        throw new IOException(e.getMessage());
    }

    return null;
}

@Override
public byte[] sendRaw(String cri, byte[] message) throws IOException {

    byte[] msg;
    byte[] tmp;

    try {
        msg = new byte[message.length + 8];
        System.arraycopy(message, 0, msg, 8, message.length);

        msg = sendOLA(cri, msg);

        if(msg != null) {
            tmp = new byte[msg.length - 8];
            System.arraycopy(msg, 8, tmp, 0, msg.length - 8);

            return tmp;
        }
    } catch(Exception e) {
        System.out.println("ALCSConnectionBean.sendRaw() : Error sending/receiving
data");
        e.printStackTrace();
        throw new IOException(e.getMessage());
    }

    return null;
}

/**
 * Send a message to the OLA and return the response.
 * @param cri the terminal address to which to send the message.

```

```

    * @param message the message to be sent, formatted in the codepage suitable for ALCS
and must also containing the EOM (end of message) character.
    * @return the received message, as received from ALCS
    * @throws IOException if there was an error sending/receiving
    */
private byte[] sendOLA(String cri, byte[] message) throws IOException {

    byte[] resp = null;

    try {

        Connection c = cf.getConnection(csi);
        Interaction i = c.createInteraction();

        message[0] = (byte)0xFF;
        message[1] = (byte)0xFF;
        message[2] = (byte)0xFF;
        message[3] = (byte)0xFF;
        message[4] = (byte)0xFF;

        int tmp = Integer.parseInt(cri, 16);
        message[5] = (byte)((tmp >> 16) & 0xff);
        message[6] = (byte)((tmp >> 8) & 0xff);
        message[7] = (byte)(tmp & 0xff);

        IndexedRecordImpl iri = new IndexedRecordImpl();
        iri.add(message);

        iri = (IndexedRecordImpl)i.execute(isi, iri);

        if((iri != null) && (iri.size() > 0)) {
            resp = (byte [])iri.get(0);

            if((resp.length == 1) && (resp[0] == (byte)0xc1)) {
                // only received ACK, real message must be coming outbound, or not at
all
                resp = null;
            }

        }

        i.close();
        c.close();

    } catch(Exception e) {
        System.out.println("ALCSConnectionBean.sendOLA() : Error sending/receiving
data");
        e.printStackTrace();
        throw new IOException(e.getMessage());
    }

    return resp;

}
}

```

Utility Class DataDumper

```

package com.ibm.alcs;

/*
 * Licensed Materials - Property of IBM
 * © Copyright IBM Corp. 2009 - All Rights Reserved.
 * DISCLAIMER:
 * THE FOLLOWING [ENCLOSED] CODE IS SAMPLE CODE CREATED BY IBM
 * CORPORATION. THIS SAMPLE CODE IS NOT PART OF ANY STANDARD IBM PRODUCT
 * AND IS PROVIDED TO YOU SOLELY FOR THE PURPOSE OF ASSISTING YOU IN THE
 * DEVELOPMENT OF YOUR APPLICATIONS. THE CODE IS PROVIDED 'AS IS',
 * WITHOUT WARRANTY OF ANY KIND. IBM SHALL NOT BE LIABLE FOR ANY DAMAGES
 * ARISING OUT OF YOUR USE OF SAMPLE CODE, EVEN IF THEY HAVE BEEN
 * ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
 */

public class DataDumper {

    /**
     * Utility function to format the provided byte array in a hex-dump like format in a
     string.
     * @param data the data to format.
     * @return the string containing the hex-dump of the data.
     */
    public static String formatData(byte[] data) {

        int i;
        int b;
        int len = data.length;
        int chars = 0;
        String indent = "    ";
        StringBuffer ascii = new StringBuffer();
        StringBuffer output = new StringBuffer();

        output.append(indent);

        for (i = 0; i < len; i++) {

            b = data[i] & 0xff;

            if((b < 0x20) || (b > 0x7e)) {
                ascii.append('.');
            }
            else {
                ascii.append((char)b);
            }
            if(b < 0x10) {
                output.append("0");
            }
            output.append(Integer.toHexString(b) + " ");
            chars++;

            if(chars == 16) {

                output.append("    ");
                output.append(ascii);
                output.append("\n");
            }
        }
    }
}

```

```

        output.append(indent);

        ascii = new StringBuffer();
        chars = 0;

    }
}

while(chars < 16) {
    output.append("  ");
    chars++;
}
output.append("  ");
output.append(ascii);
output.append("\n");

return output.toString();

}

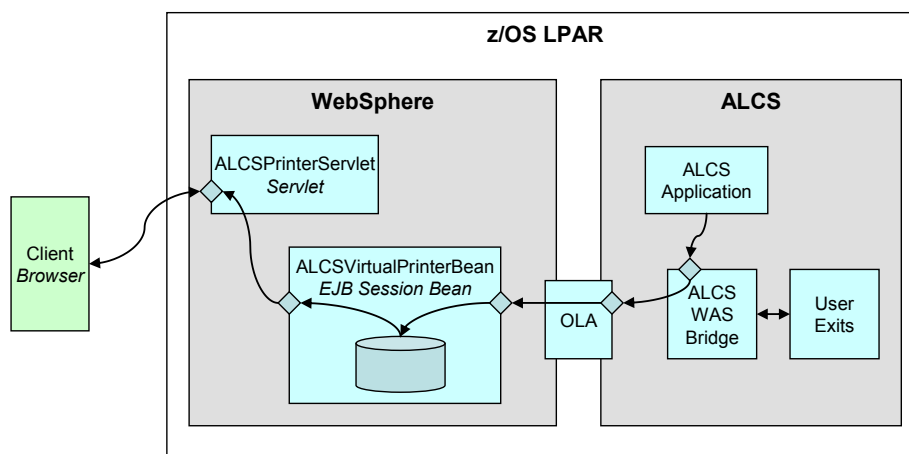
}

```

Sample Virtual Printer Application

Overview

The following diagram shows the design of the sample application. This application simulates an ALCS printer and stores the messages in an in-memory database to be retrieved later through a browser interface. The corresponding source code is included below.

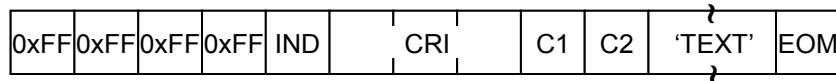


ALCS will invoke the *execute()* method of the *ALCSVirtualPrinterBean* EJB whenever there is a printer message to be sent. The *ALCSVirtualPrinterBean* will concatenate individual messages based on the end of message characters and will store complete printer messages in a trivial in-memory database (a simple Vector object).

On the user interface side the *ALCSPrinterServlet* is used to retrieve the stored messages and to display them to the user.

Message Format and Correlator

For outbound messages to WAS the correlator starts with four bytes with a fixed value of 0xFF. It is followed by a one byte value which indicates the type of printer (0x02 = ALC, 0x04 = 3270). Following that is the three byte hex value of the CRI that is the origin of this message. The CRI value is not translated between EBCDIC and ASCII. This concludes the 8 byte correlator. The following two bytes represent the standard C1/C2 control characters that indicate the output behavior of the message. The actual payload message follows and is finally terminated with one of the End of Message characters.



Code Listings

ALCS User Exit DXCUWAS3

See the corresponding section for the previous Sample application

ALCS User Exit DXCUWAS5

See the corresponding section for the previous Sample application

WebSphere Servlet ALCSPrinterServlet

```
package com.ibm.alcs;

/*
 * Licensed Materials - Property of IBM
 * © Copyright IBM Corp. 2009 - All Rights Reserved.
 * DISCLAIMER:
 * THE FOLLOWING [ENCLOSED] CODE IS SAMPLE CODE CREATED BY IBM
 * CORPORATION. THIS SAMPLE CODE IS NOT PART OF ANY STANDARD IBM PRODUCT
 * AND IS PROVIDED TO YOU SOLELY FOR THE PURPOSE OF ASSISTING YOU IN THE
 * DEVELOPMENT OF YOUR APPLICATIONS. THE CODE IS PROVIDED 'AS IS',
 * WITHOUT WARRANTY OF ANY KIND. IBM SHALL NOT BE LIABLE FOR ANY DAMAGES
 * ARISING OUT OF YOUR USE OF SAMPLE CODE, EVEN IF THEY HAVE BEEN
 * ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
 */

import java.io.IOException;
import java.io.PrintWriter;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Enumeration;
import java.util.TimeZone;
```

```

import javax.ejb.EJB;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class ALCSPrinterServlet
 */
public class ALCSPrinterServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    @EJB
    private ALCSVirtualPrinter alcs;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public ALCSPrinterServlet() {
        super();
    }

    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {

        PrintWriter out = response.getWriter();
        String cmd = request.getParameter("cmd");
        SimpleDateFormat df = new SimpleDateFormat("MM/dd/yyyy HH:mm:ssZ");
        df.setTimeZone(TimeZone.getTimeZone("America/New_York"));

        response.setContentType("text/html");

        out.println("<html>");
        out.println("<head><title>ALCS Printer</title></head>");
        out.println("<body>");

        if(cmd == null) {
            out.println("<p>The following list presents the available printers. Click on
any printer to see the stored messages.</p>");
            Enumeration<String> printers = alcs.getAvailablePrinters();
            while(printers.hasMoreElements()) {
                out.printf("<a href='ALCSPrinterServlet?cmd=msglist&cri=%1$s'>%1$s<a/><br
/>", printers.nextElement());
            }
        } else if(cmd.equalsIgnoreCase("msglist")) {
            String cri = request.getParameter("cri");

            if(cri != null) {
                out.printf("<p>These are the stored messages for printer %1$s.</p>",
cri);

                Enumeration<ALCSPrinterMessage> msgs = alcs.getPrinterMessages(cri);
                while(msgs.hasMoreElements()) {
                    ALCSPrinterMessage msg = msgs.nextElement();

```

```

        out.printf("<p style='font-family: monospace;'>Date: %1$s <br
/><pre>%2$s</pre></p><hr />", df.format(new Date(msg.getTimestamp())), msg.getMessage());
    }

    } else {
        out.println("<p>Invalid request, no printer address specified.</p>");
    }

}

out.println("</body></html>");

}

/**
 * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
 */
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
}
}

```

WebSphere EJB Session Bean Interface ALCSVirtualPrinter

```

package com.ibm.alcs;

import java.util.Enumeration;

import javax.ejb.Local;

/*
 * Licensed Materials - Property of IBM
 * © Copyright IBM Corp. 2009 - All Rights Reserved.
 * DISCLAIMER:
 * THE FOLLOWING [ENCLOSED] CODE IS SAMPLE CODE CREATED BY IBM
 * CORPORATION. THIS SAMPLE CODE IS NOT PART OF ANY STANDARD IBM PRODUCT
 * AND IS PROVIDED TO YOU SOLELY FOR THE PURPOSE OF ASSISTING YOU IN THE
 * DEVELOPMENT OF YOUR APPLICATIONS. THE CODE IS PROVIDED 'AS IS',
 * WITHOUT WARRANTY OF ANY KIND. IBM SHALL NOT BE LIABLE FOR ANY DAMAGES
 * ARISING OUT OF YOUR USE OF SAMPLE CODE, EVEN IF THEY HAVE BEEN
 * ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
 */

@Local
public interface ALCSVirtualPrinter {

    public Enumeration<String> getAvailablePrinters();

    public Enumeration<ALCSPrinterMessage> getPrinterMessages(String cri);

}

```

WebSphere EJB Session Bean ALCSVirtualPrinterBean

```
package com.ibm.alcs;

import java.util.Enumeration;
import java.util.Hashtable;
import java.util.Vector;

import javax.ejb.Local;
import javax.ejb.RemoteHome;
import javax.ejb.Stateless;

import com.ibm.websphere.ola.ExecuteHome;

/*
 * Licensed Materials - Property of IBM
 * © Copyright IBM Corp. 2009 - All Rights Reserved.
 * DISCLAIMER:
 * THE FOLLOWING [ENCLOSED] CODE IS SAMPLE CODE CREATED BY IBM
 * CORPORATION. THIS SAMPLE CODE IS NOT PART OF ANY STANDARD IBM PRODUCT
 * AND IS PROVIDED TO YOU SOLELY FOR THE PURPOSE OF ASSISTING YOU IN THE
 * DEVELOPMENT OF YOUR APPLICATIONS. THE CODE IS PROVIDED 'AS IS',
 * WITHOUT WARRANTY OF ANY KIND. IBM SHALL NOT BE LIABLE FOR ANY DAMAGES
 * ARISING OUT OF YOUR USE OF SAMPLE CODE, EVEN IF THEY HAVE BEEN
 * ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
 */

/**
 * Session Bean implementation class ALCSVirtualPrinterBean
 */
@RemoteHome(ExecuteHome.class)
@Local(ALCSVirtualPrinter.class)
@Stateless
public class ALCSVirtualPrinterBean implements ALCSVirtualPrinter {

    private static char[] HEX_CHARS = "0123456789ABCDEF".toCharArray();
    private static byte[] ALCS_ACK = {(byte)0xc1};

    private static String CODEPAGE = "Cp037";

    private static Hashtable<String, String> printBuffer = new Hashtable<String, String>();
    private static Hashtable<String, Vector<ALCSPrinterMessage>> messageStorage = new
Hashtable<String, Vector<ALCSPrinterMessage>>();

    /**
     * Default constructor.
     */
    public ALCSVirtualPrinterBean() {
    }

    /**
     * Entry point that is executed by OLA for messages originating from ALCS.
     * @param message the message received from ALCS.
     * @return the response back to ALCS, which should be the single character 'A'.
     */
    public byte[] execute(byte[] message) {
```

```

String strCRI = getCRI(message);
boolean isLastBlock = true;

System.out.println("ALCSVirtualPrinterBean.execute() : Received print message for
cri=" + strCRI);

// get the print message and translate it. It comes in as
// 8 byte correlator, 2 byte c1/c2, text, eom
try {
    String strMsg = new String(message, 10, message.length - 11, CODEPAGE);

    if(message[message.length-1] == 0x6d) {
        isLastBlock = false;
    }

    if(isLastBlock) {
        // this is the last segment
        String s = printBuffer.remove(strCRI);
        if(s != null) {
            s += strMsg;
            storeMessage(strCRI, s);
        } else {
            storeMessage(strCRI, strMsg);
        }
    } else {
        // this is not the last segment
        String s = printBuffer.remove(strCRI);
        if(s != null) {
            s += strMsg;
            printBuffer.put(strCRI, s);
        } else {
            printBuffer.put(strCRI, strMsg);
        }
    }

    return ALCS_ACK;

} catch(Exception e) {
    System.out.println("ALCSVirtualPrinterBean.execute() : Error processing incoming
message");
    e.printStackTrace();
}

return ALCS_ACK;
}

/**
 * Store message.
 * @param cri
 * @param message
 */
private void storeMessage(String cri, String message) {

    // a real world application would probably save the messages to a database

    ALCSPrinterMessage msg = new ALCSPrinterMessage(System.currentTimeMillis(), cri,
message);

```

```
Vector<ALCSPrinterMessage> list = messageStorage.get(cri);

if(list == null) {
    list = new Vector<ALCSPrinterMessage>();
    messageStorage.put(cri, list);
}

list.add(msg);
}

/**
 * Helper method to retrieve CRI in String format. This method is about 20x faster than
 * using String.format()
 * @param data the received data array
 * @return the CRI as String
 */
private String getCRI(byte[] data) {

    if((data == null) || (data.length < 8)) {
        return "";
    }

    char[] chars = new char[6];
    chars[0] = HEX_CHARS[(data[5] & 0xF0) >>> 4];
    chars[1] = HEX_CHARS[data[5] & 0x0F];
    chars[2] = HEX_CHARS[(data[6] & 0xF0) >>> 4];
    chars[3] = HEX_CHARS[data[6] & 0x0F];
    chars[4] = HEX_CHARS[(data[7] & 0xF0) >>> 4];
    chars[5] = HEX_CHARS[data[7] & 0x0F];
    return new String(chars);
}

@Override
public Enumeration<String> getAvailablePrinters() {

    return messageStorage.keys();
}

@Override
public Enumeration<ALCSPrinterMessage> getPrinterMessages(String cri) {

    Vector<ALCSPrinterMessage> list = messageStorage.get(cri);
    if(list != null) {
        return list.elements();
    }
    return null;
}
}
```

Java Class ALCSPrinterMessage

```
package com.ibm.alcs;

/*
 * Licensed Materials - Property of IBM
 * © Copyright IBM Corp. 2009 - All Rights Reserved.
 * DISCLAIMER:
 * THE FOLLOWING [ENCLOSED] CODE IS SAMPLE CODE CREATED BY IBM
 * CORPORATION. THIS SAMPLE CODE IS NOT PART OF ANY STANDARD IBM PRODUCT
 * AND IS PROVIDED TO YOU SOLELY FOR THE PURPOSE OF ASSISTING YOU IN THE
 * DEVELOPMENT OF YOUR APPLICATIONS. THE CODE IS PROVIDED 'AS IS',
 * WITHOUT WARRANTY OF ANY KIND. IBM SHALL NOT BE LIABLE FOR ANY DAMAGES
 * ARISING OUT OF YOUR USE OF SAMPLE CODE, EVEN IF THEY HAVE BEEN
 * ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
 */

public class ALCSPrinterMessage {
    private long timestamp;
    private String cri;
    private String message;

    public ALCSPrinterMessage(long timestamp, String cri, String message) {
        this.timestamp = timestamp;
        this.cri = cri;
        this.message = message;
    }

    public long getTimeStamp() {
        return timestamp;
    }

    public String getCRI() {
        return cri;
    }

    public String getMessage() {
        return message;
    }
}
```