Using Service Stream Handler

Description

The Service Stream handler allows services to use raw streams when receiving requests from clients. The `InputStream` and `OutputStream` are the only required parameters.

Developing the service

When developing a service which uses the stream handler; the best way to implement the service is by using a static java method and the `java` service engine.

The service definition

The service should always implement `serviceStreamInterface` this provides the two **IN** parameters `inputStream` and `outputStream`. As always, you can override these parameters if necessary. The only **OUT** parameter is `contentType`. This is used mainly for telling the `HttpServletResponse` object what the response type is.

services.xml

```
<services>
    <!-- Routing service to handle all incoming messages and send them to the appropriate processing service -->
    <service name="oagisMessageHandler" engine="java" transaction-timeout="300"
        location="org.ofbiz.oagis.OagisServices" invoke="oagisMessageHandler" auth="false">
        <implements service="serviceStreamInterface"/>
        <implements service="oagisMessageIdOutInterface"/>
        <attribute name="isErrorRetry" type="Boolean" mode="IN" optional="true"/>
        <override name="outputStream" optional="true"/>
    </service>

    <!-- the interface found in framework/service/servicedef/services.xml -->
    <service name="serviceStreamInterface" engine="interface">
        <description>Inteface to describe services call with streams</description>
        <attribute name="inputStream" type="java.io.InputStream" mode="IN"/>
        <attribute name="outputStream" type="java.io.OutputStream" mode="IN"/>
        <attribute name="contentType" type="String" mode="OUT"/>
    </service>
</services>
```

The service implementation

Reading the InputStream

The service implementation is simple; just like any other `Java` service. The only difference is the main parameter `inputStream`. Using a line of code:

```
InputStream in = (InputStream) context.get("inputStream");
```

You obtain a reference to the `InputStream` object. When coming in from the event handler, this is the raw data posted from the browser.

Writing to the OutputStream

The `OutputStream` in turn gets written back out to the client. This is an **IN** parameter which is referenced from the caller. Simply pull the `outputStream` from the context:

```
OutputStream out = (OutputStream) context.get("outputStream");
```

Then write to the stream as you would normally. The content type is set by the caller before the stream is closed.

NOTE: Do **NOT** close the `OutputStream` in your service; this is handled by the caller which is normally `ServiceStreamHandler.java`.

Code example

OagisServices.java

```
    public static Map oagisMessageHandler(DispatchContext ctx, Map context) {
        GenericDelegator delegator = ctx.getDelegator();
        LocalDispatcher dispatcher = ctx.getDispatcher();
        InputStream in = (InputStream) context.get("inputStream");
        List errorList = FastList.newInstance();
        Boolean isErrorRetry = (Boolean) context.get("isErrorRetry");

        GenericValue userLogin = null;
        try {
            userLogin = delegator.findByPrimaryKey("UserLogin", UtilMisc.toMap("userLoginId", "system"));
        } catch (GenericEntityException e) {
            String errMsg = "Error Getting UserLogin with userLoginId system: "+e.toString();
            Debug.logError(e, errMsg, module);
        }

        Document doc = null;
        String xmlText = null;
        try {
            BufferedReader br = new BufferedReader(new InputStreamReader(in, "UTF-8"));
            StringBuffer xmlTextBuf = new StringBuffer();
            String currentLine = null;
            while ((currentLine = br.readLine()) != null) {
                xmlTextBuf.append(currentLine);
                xmlTextBuf.append('\n');
            }
            xmlText = xmlTextBuf.toString();

            // DEJ20070804 adding this temporarily for debugging, should be changed to verbose at some point in
the future
            Debug.logWarning("Received OAGIS XML message, here is the text: \n" + xmlText, module);

            ByteArrayInputStream bis = new ByteArrayInputStream(xmlText.getBytes("UTF-8"));
            doc = UtilXml.readXmlDocument(bis, true, "OagisMessage");
        } catch (SAXException e) {
            String errMsg = "XML Error parsing the Received Message [" + e.toString() +
                "]; The text received we could not parse is: [" + xmlText + "]";
            errorList.add(UtilMisc.toMap("description", errMsg, "reasonCode", "SAXException"));
            Debug.logError(e, errMsg, module);
        } catch (ParserConfigurationException e) {
            String errMsg = "Parser Configuration Error parsing the Received Message: " + e.toString();
            errorList.add(UtilMisc.toMap("description", errMsg, "reasonCode", "ParserConfigurationException"));
            Debug.logError(e, errMsg, module);
        } catch (IOException e) {
            String errMsg = "IO Error parsing the Received Message: " + e.toString();
            errorList.add(UtilMisc.toMap("description", errMsg, "reasonCode", "IOException"));
            Debug.logError(e, errMsg, module);
        }

        if (UtilValidate.isNotEmpty(errorList)) {
            return ServiceUtil.returnError("Unable to parse received message");
        }

        Element rootElement = doc.getDocumentElement();
        rootElement.normalize();
        Element controlAreaElement = UtilXml.firstChildElement(rootElement, "os:CNTROLAREA");
        Element bsrElement = UtilXml.firstChildElement(controlAreaElement, "os:BSR");
        String bsrVerb = UtilXml.childElementValue(bsrElement, "of:VERB");
        String bsrNoun = UtilXml.childElementValue(bsrElement, "of:NOUN");

        Element senderElement = UtilXml.firstChildElement(controlAreaElement, "os:SENDER");
        String logicalId = UtilXml.childElementValue(senderElement, "of:LOGICALID");
        String component = UtilXml.childElementValue(senderElement, "of:COMPONENT");
```

```
        String task = UtilXml.childElementValue(senderElement, "of:TASK");
        String referenceId = UtilXml.childElementValue(senderElement, "of:REFERENCEID");

        if (UtilValidate.isEmpty(bsrVerb) || UtilValidate.isEmpty(bsrNoun)) {
            return ServiceUtil.returnError("Was able to receive and parse the XML message, but BSR->NOUN [" +
bsrNoun +
                "] and/or BSR->VERB [" + bsrVerb + "] are empty");
        }

        GenericValue oagisMessageInfo = null;
        Map oagisMessageInfoKey = UtilMisc.toMap("logicalId", logicalId, "component", component, "task", task,
"referenceId", referenceId);
        try {
            oagisMessageInfo = delegator.findByPrimaryKey("OagisMessageInfo", oagisMessageInfoKey);
        } catch (GenericEntityException e) {
            String errMsg = "Error Getting Entity OagisMessageInfo: " + e.toString();
            Debug.logError(e, errMsg, module);
        }

        Map messageProcessContext = UtilMisc.toMap("document", doc, "userLogin", userLogin);

        // call async, no additional results to return: Map subServiceResult = FastMap.newInstance();
        if (UtilValidate.isNotEmpty(oagisMessageInfo)) {
            if (Boolean.TRUE.equals(isErrorRetry) || "OAGMP_SYS_ERROR".equals(oagisMessageInfo.getString
("processingStatusId"))) {
                // there was an error last time, tell the service this is a retry
                messageProcessContext.put("isErrorRetry", Boolean.TRUE);
            } else {
                String responseMsg = "Message already received with ID: " + oagisMessageInfoKey;
                Debug.logError(responseMsg, module);

                List errorMapList = UtilMisc.toList(UtilMisc.toMap("reasonCode", "MessageAlreadyReceived",
"description", responseMsg));

                Map sendConfirmBodCtx = FastMap.newInstance();
                sendConfirmBodCtx.put("logicalId", logicalId);
                sendConfirmBodCtx.put("component", component);
                sendConfirmBodCtx.put("task", task);
                sendConfirmBodCtx.put("referenceId", referenceId);
                sendConfirmBodCtx.put("errorMapList", errorMapList);
                sendConfirmBodCtx.put("userLogin", userLogin);

                try {
                    // run async because this will send a message back to the other server and may take some
time, and/or fail
                    dispatcher.runAsync("oagisSendConfirmBod", sendConfirmBodCtx, null, true, 60, true);
                } catch (GenericServiceException e) {
                    String errMsg = "Error sending Confirm BOD: " + e.toString();
                    Debug.logError(e, errMsg, module);
                }
                Map result = ServiceUtil.returnSuccess(responseMsg);
                result.put("contentType", "text/plain");
                return result;
            }
        }

        Debug.logInfo("Processing OAGIS message with verb [" + bsrVerb + "] and noun [" +
            bsrNoun + "] with context: " + messageProcessContext, module);

        if (bsrVerb.equalsIgnoreCase("CONFIRM") && bsrNoun.equalsIgnoreCase("BOD")) {
            try {
                // subServiceResult = dispatcher.runSync("oagisReceiveConfirmBod", messageProcessContext);
                dispatcher.runAsync("oagisReceiveConfirmBod", messageProcessContext, true);
            } catch (GenericServiceException e) {
                String errMsg = "Error running service oagisReceiveConfirmBod: " + e.toString();
                errorList.add(UtilMisc.toMap("description", errMsg, "reasonCode", "GenericServiceException"));
                Debug.logError(e, errMsg, module);
            }
        } else if (bsrVerb.equalsIgnoreCase("SHOW") && bsrNoun.equalsIgnoreCase("SHIPMENT")) {
            try {
                //subServiceResult = dispatcher.runSync("oagisReceiveShowShipment", messageProcessContext);
```

```
                // DEJ20070808 changed to run asynchronously and persisted so that if it fails it will retry;
                // for transaction deadlock and other reasons
                dispatcher.runAsync("oagisReceiveShowShipment", messageProcessContext, true);
            } catch (GenericServiceException e) {
                String errMsg = "Error running service oagisReceiveShowShipment: " + e.toString();
                errorList.add(UtilMisc.toMap("description", errMsg, "reasonCode", "GenericServiceException"));
                Debug.logError(e, errMsg, module);
            }
        } else if (bsrVerb.equalsIgnoreCase("SYNC") && bsrNoun.equalsIgnoreCase("INVENTORY")) {
            try {
                //subServiceResult = dispatcher.runSync("oagisReceiveSyncInventory", messageProcessContext);
                // DEJ20070808 changed to run asynchronously and persisted so that if it fails it will retry;
                // for transaction deadlock and other reasons
                dispatcher.runAsync("oagisReceiveSyncInventory", messageProcessContext, true);
            } catch (GenericServiceException e) {
                String errMsg = "Error running service oagisReceiveSyncInventory: " + e.toString();
                errorList.add(UtilMisc.toMap("description", errMsg, "reasonCode", "GenericServiceException"));
                Debug.logError(e, errMsg, module);
            }
        } else if (bsrVerb.equalsIgnoreCase("ACKNOWLEDGE") && bsrNoun.equalsIgnoreCase("DELIVERY")) {
            Element dataAreaElement = UtilXml.firstChildElement(rootElement, "ns:DATAAREA");
            Element ackDeliveryElement = UtilXml.firstChildElement(dataAreaElement, "ns:ACKNOWLEDGE_DELIVERY");
            Element receiptlnElement = UtilXml.firstChildElement(ackDeliveryElement, "ns:RECEIPTLN");
            Element docRefElement = UtilXml.firstChildElement(receiptlnElement, "os:DOCUMNTREF");
            String docType = docRefElement != null ? UtilXml.childElementValue(docRefElement, "of:DOCTYPE") :
null;
            String disposition = UtilXml.childElementValue(receiptlnElement, "of:DISPOSITN");

            if ("PO".equals(docType)) {
                try {
                    //subServiceResult = dispatcher.runSync("oagisReceiveAcknowledgeDeliveryPo",
messageProcessContext);
                    dispatcher.runAsync("oagisReceiveAcknowledgeDeliveryPo", messageProcessContext, true);
                } catch (GenericServiceException e) {
                    String errMsg = "Error running service oagisReceiveAcknowledgeDeliveryPo: " + e.toString();
                    errorList.add(UtilMisc.toMap("description", errMsg, "reasonCode",
"GenericServiceException"));
                    Debug.logError(e, errMsg, module);
                }
            } else if ("RMA".equals(docType)) {
                try {
                    //subServiceResult = dispatcher.runSync("oagisReceiveAcknowledgeDeliveryRma",
messageProcessContext);
                    dispatcher.runAsync("oagisReceiveAcknowledgeDeliveryRma", messageProcessContext, true);
                } catch (GenericServiceException e) {
                    String errMsg = "Error running service oagisReceiveAcknowledgeDeliveryRma: " + e.toString();
                    errorList.add(UtilMisc.toMap("description", errMsg, "reasonCode",
"GenericServiceException"));
                    Debug.logError(e, errMsg, module);
                }
            } else if (UtilValidate.isEmpty(docType) && ("NotAvailableTOAvailable".equals(disposition) ||
                       "AvailableTONotAvailable".equals(disposition))) {
                try {
                    dispatcher.runAsync("oagisReceiveAcknowledgeDeliveryStatus", messageProcessContext, true);
                } catch (GenericServiceException e) {
                    String errMsg = "Error running service oagisReceiveAcknowledgeDeliveryStatus: " + e.
toString();
                    errorList.add(UtilMisc.toMap("description", errMsg, "reasonCode",
"GenericServiceException"));
                    Debug.logError(e, errMsg, module);
                }
            } else {
                return ServiceUtil.returnError("For Acknowledge Delivery message could not determine if it is
for a PO " +
                    "or RMA or Status Change. DOCTYPE from message is [" + docType + "], DISPOSITN is [" +
disposition + "]");
            }
        } else {
            String errMsg = "Unknown Message Type Received, verb/noun combination not supported: verb=[" +
bsrVerb + "], noun=[" + bsrNoun + "]";
            Debug.logError(errMsg, module);
```

```
            return ServiceUtil.returnError(errMsg);
        }

        Map result = ServiceUtil.returnSuccess();
        result.put("contentType", "text/plain");

        /* no sub-service error processing to be done here, all handled in the sub-services:
        result.putAll(subServiceResult);
        List errorMapList = (List) subServiceResult.get("errorMapList");
        if (UtilValidate.isNotEmpty(errorList)) {
            Iterator errListItr = errorList.iterator();
            while (errListItr.hasNext()) {
                Map errorMap = (Map) errListItr.next();
                errorMapList.add(UtilMisc.toMap("description", errorMap.get("description"), "reasonCode",
errorMap.get("reasonCode")));
            }
            result.put("errorMapList", errorMapList);
        }
        */

        return result;
    }
```

Configuring the controller entry

1. Be sure the `ServiceStreamHandler` is set as one of the event handlers.
2. Configure your events (mount points) to use this handler.

controller.xml

```
    <handler name="stream" type="request" class="org.ofbiz.webapp.event.ServiceStreamHandler"/>
    ...
    <request-map uri="oagisMessageHandler">
        <security https="true" auth="false" cert="true"/>
        <event type="stream" invoke="oagisMessageHandler"/>
        <response name="success" type="none"/>
    </request-map>
```