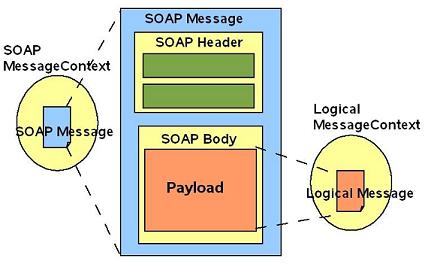
**Handlers in JAX-WS**

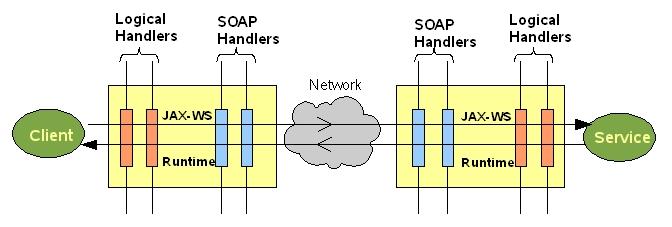
Handlers are message interceptors that can be easily plugged in to the JAX-WS runtime to do additional processing of the inbound and outbound messages. JAX-WS defines two types of handlers, logical handlers and protocol handlers. Protocol handlers are specific to a protocol and may access or change the protocol specific aspects of a message. Logical handlers are protocol-agnostic and cannot change any protocol-specific parts (like headers) of a message. Logical handlers act only on the payload of the message.

Handlers are invoked with a message context that provides methods to access and modify inbound and outbound messages and to manage a set of properties. The, message context properties can be used to communicate information between handlers and client and service implementations. SOAP handlers should extend javax.xml.ws.handler.soap.SOAPHandler, which is defined for SOAP binding by JAX-WS specification. SOAP handlers are invoked with SOAPMessageContext, which provides methods to access SOAPMessage. SOAPMessageContext.getMessage() gives a SOAPMessage. One can use the SAAJ API to manipulate the SOAP Message.

Logical handlers extend javax.xml.ws.handler.LogicalHandler and provide access to message context and message payload. If you are using SOAP over HTTP, the content of the SOAP body forms the payload. If you are using XML over HTTP, the XML content of the primary part of the message becomes the payload. Logical handlers are invoked with LogicalMessageContext. LogicalMessageContext.getMessage() returns a LogicalMessage. The LogicalMessage represents a protocol neutral XML message and contains methods that provide access to the payload of the message.



Logical handlers can coexist with SOAP handlers in a handler chain. During runtime, the handler chain is re-ordered such that logical handlers are executed before the SOAP handlers on an outbound message and SOAP handlers are executed before logical handlers on an inbound message. The following figure shows how logical and SOAP handlers are invoked during a request and response.

  
**Writing a Handler in JAX-WS:**

Writing a handler in JAX-WS is easy. A basic handler should implement the following three methods.

*handleMessage( )*: This is called for inbound and outbound messages.

*handleFault( )*: This is called instead of handleMessage( ), when the message contains a protocol fault.

*close( )*: This is called after the completion of message processing by all handlers for each web service invocation (after completion of MEP). This can be useful to clean up any resources used during processing the message.

The following section describes how to SOAP handler and logical handler.

**Writing a SOAP Handler:**

Writing a SOAP handler involves extending javax.xml.ws.handler.soap.SOAPHandler. A sample SOAP logging handler can be accessed [here](https://jax-ws.java.net/articles/handlers_introduction/SOAPLoggingHandler.java). The following snippet shows how you can access SOAP Message in a SOAP handler.

public boolean handleMessage(SOAPMessageContext smc) {

Boolean outboundProperty = (Boolean) smc.get (MessageContext.MESSAGE\_OUTBOUND\_PROPERTY);

if (outboundProperty.booleanValue()) {

out.println("\nOutbound message:");

} else {

out.println("\nInbound message:");

}

SOAPMessage message = smc.getMessage();

// Use SAAJ API to manipulate the SOAP Message

try {

message.writeTo(out);

out.println(""); // just to add a newline

} catch (Exception e) {

out.println("Exception in handler: " + e);

}

}

**Writing a Logical Handler:**

Writing a logical handler involves extending javax.xml.ws.handler.LogicalHandler. A sample logical logging handler can be accessed [here](https://jax-ws.java.net/articles/handlers_introduction/LogicalLoggingHandler.java). The following snippet shows how you can access and manipulate a message payload from message context in a logical handler. LogicalMessage.getPayload() returns the payload as Source . The type of source returned depends on the JAX-WS runtime. If the returned source is DOMSource, then modifications to the encapsulated DOM tree change the message payload in-place, there is no need to subsequently call setPayload(). Other types of Source provide only read access to the message payload and require setPayload() for modifications.

public boolean handleMessage(LogicalMessageContext context) {

Boolean outboundProperty = (Boolean) context.get(MessageContext.MESSAGE\_OUTBOUND\_PROPERTY);

if (outboundProperty) {

out.println("\nOutbound message:");

} else {

out.println("\nInbound message:");

}

LogicalMessage lm = context.getMessage();

Source payload = lm.getPayload();

// Process Payload Source

printSource(payload);

// ....

// If the payload is modified, Do lm.setPayload(source) to be safe,

// Without it, behavior may vary on the kind of source returned in lm.getPayload().

// See LogicalMessage JavaDoc for more details.

// lm.setPayload(modifiedPayload);

return true;

}

You can also pass JAXBContext, to get the payload as JAXB object, as shown below. Note that there is no connection between the returned object and the message payload, changes to the payload require calling setPayload().

LogicalMessage lm = context.getMessage();

Object jaxbObject = lm.getPayload(jaxbContext);

// Modify JAXB Object

lm.setPayload(modifiedJaxbObject,jaxbContext);

**Summary:**

The following table summarizes the differences between a SOAP handler and logical handler.

|  | **Logical handler** | **SOAP handler** |
| --- | --- | --- |
| *extends* | javax.xml.ws.handler.LogicalHandler | javax.xml.ws.handler.soap.SOAPHandler |
| *message context* | javax.xml.ws.handler.LogicalMessageContext | javax.xml.ws.handler.soap.SOAPMessageContext |
| *getMessage() of message context gives* | javax.xml.ws.LogicalMessage | javax.xml.soap.SOAPMessage |

SOAP handlers are generally used to process SOAP-specific information like SOAP headers. For example, a SOAP Handler can process security headers in a message and pass the request to the endpoint if the message has the required credentials. Logical handlers are commonly used, if the processing does not need access to SOAP headers, for validation of the payload, and with REST style Web Services etc. If you have the JAXBContext and want to alter something in the message, it's very simple with a logical handler as you can get the jaxb objects and call Java methods on them rather than dealing with the Source or the SOAPMessage using SAAJ API in a SOAP handler.

.

\*\*\*\*\*