

```

Name listingElementName = soapFactory.createName(
    "propertyListing", "realProperty",
    "http://schemas.realhouses.com/listingSubmission");
SOAPBodyElement listingElement =
body.addBodyElement(listingElementName);

Name attname = soapFactory.createName("id");
listingElement.addAttribute(attname, "property_1234");

SOAPElement listingAgency =
listingElement.addChildElement("listingAgency");
listingAgency.addTextNode("Really Nice Homes, Inc");

SOAPElement listingType =
listingElement.addChildElement("listingType");
listingType.addTextNode("add");

SOAPElement propertyAddress =
listingElement.addChildElement("propertyAddress");
SOAPElement street = propertyAddress.addChildElement("street");
street.addTextNode("1234 Main St");
SOAPElement city = propertyAddress.addChildElement("city");
city.addTextNode("Pleasantville");
SOAPElement state = propertyAddress.addChildElement("state");
state.addTextNode("CA");
SOAPElement zip = propertyAddress.addChildElement("zip");
zip.addTextNode("94521");

SOAPElement listPrice =
listingElement.addChildElement("listPrice");
listPrice.addTextNode("25000");

```

Note you add the property's unique ID as an attribute to the `propertyListing` element. Further, you qualify the `propertyListing` element with a `QName`, or namespace-aware name.

You can add attachments to the SOAP message in several ways. In this example, you first create elements to denote the listed property's front and interior images. Each has an `href` attribute designating the attachment's content ID:

```

String frontImageID = "property1234_front_jpeg@realhouses.com";
SOAPElement frontImRef =
listingElement.addChildElement("frontImage");
Name hrefAttName = soapFactory.createName("href");
frontImRef.addAttribute(hrefAttName, frontImageID);

String interiorID = "property1234_interior_jpeg@realhouses.com";
SOAPElement interiorImRef =
listingElement.addChildElement("interiorImage");
interiorImRef.addAttribute(hrefAttName, interiorID);

```

To easily attach the required image files to the message, use a `javax.activation.DataHandler` object from the JavaBeans Activation Framework. `DataHandler` can automatically detect the data type passed to it, and it can therefore automatically assign the appropriate MIME content type to the attachment:

```
URL url = new URL("file:///export/files/pic1.jpg");
DataHandler dataHandler = new DataHandler(url);
AttachmentPart att = message.createAttachmentPart(dataHandler);
att.setContentId(frontImageID);
message.addAttachmentPart(att);
```

Alternatively, you may be able to pass an `Object`, along with the correct MIME type, to `createAttachmentPart()`. That method resembles the first one. Internally, the SAAJ implementation will likely look for a `DataContentHandler` to handle the specified MIME type. If it can't find a suitable handler, `createAttachmentPart()` will throw an `IllegalArgumentException`:

```
URL url2 = new URL("file:///export/files/pic2.jpg");
Image im = Toolkit.getDefaultToolkit().createImage(url2);
AttachmentPart att2 = message.createAttachmentPart(im,
"image/jpeg");
att2.setContentId(interiorID);
message.addAttachmentPart(att2);
```