7.8 MultiPart

MultiPart

Apache Wink provides a MultiPart data model and providers for producing and consuming multipart messages (multipart/*). All of the model classes are located under the org.apache.wink.common.model.multipart package distributed with the wink-common jar.

The data model can be used with the wink-server module or with the wink-client module.

	Supported	Media Types	Data Model	Provider registration
Read	Yes	multipart/*	org.apache.wink .common.model .multipart.InMultiPart org.apache.wink .common.model .multipart .BufferedInMultiPart	Not required. Registered by default
Write	Yes	multipart/*	org.apache.wink .common.model .multipart.OutMultiPart org.apache.wink .common.model .multipart .BufferedOutMultiPart	Not required. Registered by default

Serialization and De-serialization

The serialization and de-serialization of a multipart message is performed by the multipart providers. The serialization and de-serialization of the parts that make up the multipart message is performed as if each part is a separate message and in accordance with the JAX-RS specification. This means that every part is serialized and de-serialized using the appropriate provider that matches the binding class and content media type of that specific part.

Main Classes

The multipart data model is comprised of the following main classes:

- **InMultiPart** - is used for de-serialization of an incoming multipart message.
- **InPart** - represents a single part contained in an incoming multipart message.
- **OutMultiPart** - is used for serialization of an outgoing multipart message.
- **OutPart** - represents a single part contained in an outgoing multipart message.

Streaming Multipart

The base multipart classes are designed to handle multipart messages without buffering the data in order to avoid possible memory issues. This means that the data is accessible only once by the use of an iterator.

Buffering Multipart

The BufferedInMultiPart and BufferedOutMultiPart classes are used to handle multipart messages where the complete message is buffered in the memory, allowing random and multiple access of the data. These classes are suitable for situations where the multipart message is small.

Examples

The following examples illustrate the usage of the multipart data model.

Multipart Consumption

The following example illustrates the usage of a streaming multipart message.

```
@Path("files")
@POST
@Produces( MediaType.TEXT_PLAIN)
@Consumes( MediaTypeUtils.MULTIPART_FORM_DATA)
public String uploadFiles(InMultiPart inMP) throws IOException {
    while (inMP.hasNext()) {
        InPart part = inMP.next();
        MultivaluedMap<String, String> heades = part.getHeaders();
        String CDHeader = heades.getFirst("Content-Disposition");
        InputStream is = part.getBody(InputStream.class, null);
        // use the input stream to read the part body
    }
}
```

* Detailed example of the MultiPart implementation can be seen at the MultiPart example.

Buffered Multipart Consumption

The following example illustrates the usage of a buffering multipart message.

```
@Path("users")
@POST
@Consumes( {"multipart/mixed"})
public BufferedOutMultiPart addUsers(BufferedInMultiPart inMP) throws IOException {
    List<InPart> parts = inMP.getParts();
    for (InPart p : parts) {
        User u = p.getBody(User.class, null);
        // use the user object retrieved from the part body

    }
}
```

* Detailed example of the MultiPart implementation can be seen at the MultiPart example.