# WSDL EJB Extension

## 1 Details

The EJB binding is a WSDL binding that allows abstract functionality in the abstract service description (messages, operations and port types) to be mapped to functionality offered by an enterprise java bean directly. This means that an EJB can be described using WSDL, and can be accessed as a WSDL-described service through WSIF.

The EJB binding extends WSDL with the following extensibility elements:

```
<definitions .... >

    <!-- EJB binding -->
    <binding ... >
        <ejb:binding/>
        <format:typeMapping style="uri" encoding="..."/>?
            <format:typeMap typeName="qname"|elementName="qname"
formatType="nmtoken"/>*
        </format:typeMapping>
        <operation>*
            <ejb:operation
                methodName="nmtoken"
                parameterOrder="nmtoken"?
                returnPart="nmtoken"?
                interface="home|remote"? />?
            <input name="nmtoken"? />?
            <output name="nmtoken"? />?
            <fault name="nmtoken"? />?
        </operation>
    </binding>

    <service ... >
        <port>*
             <ejb:address
                 className="nmtoken"
                 jndiName="nmtoken"?
                 initialContextFactory="nmtoken"?
                 jndiProviderURL="url"?  />
        </port>
    </service>
</definitions>
```

Each element is described in detail below.

- **`ejb:binding`** This indicates that the binding is an EJB binding.

- **`format:typemapping`** For information about the format:typeMapping and format:typeMap please see Format Type Mapping.
- **`ejb:operation`** This element maps an abstract WSDL operation to a method offered by an EJB interface. The *methodName* attribute specifies the name of the java method corresponding to the abstract operation. The *parameterOrder* attribute is similar to and overrides the *parameterOrder* specification in the abstract operation. It specifies the ordering of the input message parts for the invocation; in the EJB binding case it identifies the method signature. Having a *parameterOrder* attribute here allows us to map an abstract operation to an EJB method even if their signatures aren't compatible in the ordering of parts. The *returnPart* is that part of the abstract output message which corresponds to the return value of the java method. The *ejbInterface* attribute specifies whether the method is offered through the home interface or the remote interface of the EJB; by default the client will assume that it is a method on the remote interface.
- **`ejb:address`** This element is an extension under the WSDL *port* element that allows specification of an EJB object as an endpoint for a service available via the EJB binding. The port whose address is specified this way must be associated with an EJB binding only.
  The *className* attribute specifies the fully qualified name of the home interface class of the EJB. The *jndiName* attribute specifies the name under which this EJB can be looked up in a JNDI context. The optional *initialContextFactory* and *jndiProviderURL* attributes complete the set if information required to perform a JNDI lookup for the EJB. It is up to the service provider to insure that all java methods used for mapping abstract operations must be publicly available through the specified interface in the EJB.

## 2 Example

```
<?xml version="1.0" ?>

<definitions targetNamespace="http://wsifservice.addressbook/"
             xmlns:tns="http://wsifservice.addressbook/"
             xmlns:typens="http://wsiftypes.addressbook/"
             xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
             xmlns:format="http://schemas.xmlsoap.org/wsdl/formatbinding/"
             xmlns:java="http://schemas.xmlsoap.org/wsdl/java/"
             xmlns="http://schemas.xmlsoap.org/wsdl/">

  <!-- type defs -->
  <types>
    <xsd:schema
      targetNamespace="http://wsiftypes.addressbook/"
                xmlns:xsd="http://www.w3.org/1999/XMLSchema">
      <xsd:complexType name="phone">
        <xsd:element name="areaCode" type="xsd:int"/>
        <xsd:element name="exchange" type="xsd:string"/>
        <xsd:element name="number" type="xsd:string"/>
```

```
      </xsd:complexType>

      <xsd:complexType name="address">
        <xsd:element name="streetNum" type="xsd:int"/>
        <xsd:element name="streetName" type="xsd:string"/>
        <xsd:element name="city" type="xsd:string"/>
        <xsd:element name="state" type="xsd:string"/>
        <xsd:element name="zip" type="xsd:int"/>
        <xsd:element name="phoneNumber" type="typens:phone"/>
      </xsd:complexType>
    </xsd:schema>
  </types>

  <!-- message declns -->
  <message name="AddEntryWholeNameRequestMessage">
    <part name="name" type="xsd:string"/>
    <part name="address" type="typens:address"/>
  </message>

  <message name="AddEntryFirstAndLastNamesRequestMessage">
    <part name="firstName" type="xsd:string"/>
    <part name="lastName" type="xsd:string"/>
    <part name="address" type="typens:address"/>
  </message>

  <message name="GetAddressFromNameRequestMessage">
    <part name="name" type="xsd:string"/>
  </message>

  <message name="GetAddressFromNameResponseMessage">
    <part name="address" type="typens:address"/>
  </message>

  <!-- port type declns -->
  <portType name="AddressBook">
    <operation name="addEntry">
      <input name="AddEntryWholeNameRequest"
message="tns:AddEntryWholeNameRequestMessage"/>
    </operation>
    <operation name="addEntry">
      <input name="AddEntryFirstAndLastNamesRequest"
message="tns:AddEntryFirstAndLastNamesRequestMessage"/>
    </operation>
    <operation name="getAddressFromName">
      <input name="GetAddressFromNameRequest"
message="tns:GetAddressFromNameRequestMessage"/>
      <output name="GetAddressFromNameResponse"
message="tns:GetAddressFromNameResponseMessage"/>
    </operation>
  </portType>

  <!-- binding declns -->
  <binding name="EJBBinding" type="tns:AddressBook">
    <ejb:binding/>
```

```
    <format:typeMapping encoding="Java" style="Java">
      <format:typeMap typeName="typens:address"
formatType="addressbook.wsiftypes.Address" />
      <format:typeMap typeName="xsd:string" formatType="java.lang.String"
/>
    </format:typeMapping>
    <operation name="addEntry">
      <ejb:operation
         methodName="addEntry"
         parameterOrder="name address"
         interface="remote" />
      <input name="AddEntryWholeNameRequest"/>
    </operation>
    <operation name="addEntry">
      <ejb:operation
         methodName="addEntry"
         parameterOrder="firstName lastName address"
         interface="remote" />
      <input name="AddEntryFirstAndLastNamesRequest"/>
    </operation>
    <operation name="getAddressFromName">
      <ejb:operation
         methodName="getAddressFromName"
         parameterOrder="name"
         interface="remote"
         returnPart="address" />
      <input name="GetAddressFromNameRequest"/>
      <output name="GetAddressFromNameResponse"/>
    </operation>
  </binding>

  <!-- service decln -->
  <service name="AddressBookService">
    <port name="EJBPort" binding="tns:EJBBinding">
      <ejb:address className="addressbook.wsiftypes.AddressBookHome"
                   jndiName="/services/addressbook"
initialContextFactory="com.mycompany.server.MyappInitialContext"
jndiProviderURL="ormi://myserver.mycompany.com/ejbsample"/>
    </port>
  </service>

</definitions>
```

In the above example, an address book service is offered through an EJB. The EJB object is looked up through the specified context factory using the provider URL and lookup name. The service offers two variants of the `addEntry` operation. One takes an input message consisting of a full name and address and the other accepts an input message with a first name and last name (as separate message parts) and an address. Both of these operations are mapped to a java method called `addEntry`, but there are different parameter lists (this is an overloaded method in the implementing class). Finally, the abstract operation `getAddressFromName` is mapped to a java method of the same name. All of these methods are offered through the remote interface of the EJB. The types

`typesns:address` and `xsd:string` used during method invocations are mapped to java types `addressbook.wsiftypes.Address` and `java.lang.String` respectively. Note that the type `typesns:phone` does not need to be mapped to a java type since it is contained within the address type which is represented by a known java class. WSIF does not need to know how this class represents the information described in the schema type if that is being done is service invocation. If we needed to inspect the information in the WSIF message and transform it in some way, we would need details on how the java class represents the type information (see the above discussion on the java encoding).