



# JAX-RPC Developers Tutorial

---

Web Services Reference Implementation  
Version 1.0

Java™ 2 Platform, Micro Edition

Sun Microsystems, Inc.  
4150 Network Circle  
Santa Clara, California 95054  
U.S.A. 650-960-1300

October 2003

Copyright © 2003 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, U.S.A. All rights reserved.

Sun Microsystems, Inc. has intellectual property rights relating to technology embodied in the product that is described in this document. In particular, and without limitation, these intellectual property rights may include one or more of the U.S. patents listed at <http://www.sun.com/patents> and one or more additional patents or pending patent applications in the U.S. and in other countries.

This document and the product to which it pertains are distributed under licenses restricting their use, copying, distribution, and decompilation. No part of the product or of this document may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any.

Third-party software, including font technology, is copyrighted and licensed from Sun suppliers.

Sun; Sun Microsystems; the Sun logo; Solaris; Java; J2ME; J2SE; J2EE; JCP; Java 2 Platform, Micro Edition; Java 2 Platform, Standard Edition; Java 2 Platform, Enterprise Edition; Java API for XML Processing; Java API for XML-Based RPC; Java Developer Connection; and Java Community Process are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

The Adobe® logo is a registered trademark of Adobe Systems, Incorporated.

Federal Acquisitions: Commercial Software - Government Users Subject to Standard License Terms and Conditions.

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

---

Copyright © 2003 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, Etats-Unis. Tous droits réservés.

Sun Microsystems, Inc. a les droits de propriété intellectuels relatants à la technologie incorporée dans le produit qui est décrit dans ce document. En particulier, et sans la limitation, ces droits de propriété intellectuels peuvent inclure un ou plus des brevets américains énumérés à <http://www.sun.com/patents> et un ou les brevets plus supplémentaires ou les applications de brevet en attente dans les Etats - Unis et dans les autres pays.

Ce produit ou document est protégé par un copyright et distribué avec des licences qui en restreignent l'utilisation, la copie, la distribution, et la décompilation. Aucune partie de ce produit ou document ne peut être reproduite sous aucune forme, par quelque moyen que ce soit, sans l'autorisation préalable et écrite de Sun et de ses bailleurs de licence, s'il y ena.

Le logiciel détenu par des tiers, et qui comprend la technologie relative aux polices de caractères, est protégé par un copyright et licencié par des fournisseurs de Sun.

Sun; Sun Microsystems; the Sun logo; Solaris; Java; J2ME; J2SE; J2EE; JCP; Java 2 Platform, Micro Edition; Java 2 Platform, Standard Edition; Java 2 Platform, Enterprise Edition; Java API for XML Processing; Java API for XML-Based Processing; Java Developer Connection; et Java Community Process sont des marques de fabrique ou des marques déposées de Sun Microsystems, Inc. aux Etats-Unis et dans d'autres pays.

Le logo Adobe® est une marque déposée de Adobe Systems, Incorporated.

LA DOCUMENTATION EST FOURNIE "EN L'ÉTAT" ET TOUTES AUTRES CONDITIONS, DECLARATIONS ET GARANTIES EXPRESSES OU TACITES SONT FORMELLEMENT EXCLUES, DANS LA MESURE AUTORISEE PAR LA LOI APPLICABLE, Y COMPRIS NOTAMMENT TOUTE GARANTIE IMPLICITE RELATIVE A LA QUALITE MARCHANDE, A L'APTITUDE A UNE UTILISATION PARTICULIERE OU A L'ABSENCE DE CONTREFAÇON.



Please  
Recycle



Adobe PostScript

# Contents

---

## **Preface v**

### **1. Introducing the JAX-RPC Optional Package 1**

Purpose of the JAX-RPC Optional Package 1

Contents of the JAX-RPC Optional Package 2

The J2ME Client Platform 2

Difference Between J2SE and J2ME 3

Web Services Definition Language Files 3

The JAX-RPC API Subset 4

com.sun.j2mews.xml.rpc 4

javax.microedition.xml.rpc 4

javax.xml.namespace 5

javax.xml.rpc 5

The Remote Method Invocation API 5

java.rmi 5

The Service Provider Interface 6

### **2. Sample JAX-RPC Web Service 7**

The Web Services Client Model 7

The config.xml Configuration File 9

The Stub Generator 10

Generating a J2ME Web Services Stub 10

Stub Generator Output	11
The Stub	11
The Web Service WSDL File	12
WSDL File Mappings	13
Packaging Stub and Data Classes with a MIDlet	16
Making The Stub Available to An Application	16
Programming Stub Calls into An Application	16
<b>A. EmployeeDB.wsdl Sample File</b>	<b>19</b>
<b>B. EDBDMidlet2.java Sample File</b>	<b>25</b>

# Preface

This document describes the Java™ 2 Platform, Micro Edition (J2ME™) Web Services Reference Implementation. It provides useful information on the JAX-RPC optional package, including descriptions of the JAX-RPC APIs, the Remote Method Invocation (RMI) APIs, and the Service Provider Interface (SPI). It also provides a brief discussion about the Web Services Definition Language (WSDL) and the J2ME client platform.

## Typographic Conventions

Typographic Conventions

Typeface	Meaning	Examples
AaBbCc123	The names of commands, files, and directories; on-screen computer output	Edit your <code>.login</code> file. Use <code>ls -a</code> to list all files. % You have mail.
<b>AaBbCc123</b>	What you type, when contrasted with on-screen computer output	% <b>su</b> Password:
<i>AaBbCc123</i>	Book titles, new words or terms, words to be emphasized	Read Chapter 6 in the <i>User's Guide</i> . These are called <i>class</i> options. You <i>must</i> be superuser to do this.
	Command-line variable; replace with a real name or value	To delete a file, type <code>rm filename</code> .

---

## Related Documentation

Related Documentation

Application	Title
Installation	<i>J2ME Web Services Reference Implementation: Installation Guide</i>
JAXP Optional Package	<i>J2ME Web Services Reference Implementation: JAXP Developers Tutorial</i>
Specification	<i>J2ME Web Services Specification, version 1.0</i>

---

## Accessing Sun Documentation Online

The Sun Microsystems Java Developer Connection<sup>SM</sup> web site enables you to access Java<sup>TM</sup> platform technical documentation on the Web:

<http://developer.java.sun.com/developer/infodocs/>

---

## Sun Welcomes Your Comments

We are interested in improving our documentation and welcome your comments and suggestions. You can email your comments to us at:

[docs@java.sun.com](mailto:docs@java.sun.com)

# Introducing the JAX-RPC Optional Package

---

The Java 2, Micro Edition (J2ME) Web Services Reference Implementation provides two XML-based optional packages that can be used to develop Java web services for small, reduced-memory, handheld devices such as cell phones, PDAs, and pagers. These packages are:

- The Java™ API for XML Processing (JAXP)
- The Java™ API for XML-Based RPC (JAX-RPC)

This document describes the J2ME Web Services JAX-RPC optional package. For information on JAXP, see the *J2ME Web Services JAXP Developers Tutorial*.

---

## Purpose of the JAX-RPC Optional Package

The Java API for XML-Based RPC (JAX-RPC) is an implementation of Remote Procedure Call (RPC) technology in the Java language, and is part of the Java™ 2, Enterprise Edition (J2EE™) platform. The JAX-RPC optional package subset, provided with the J2ME Web Services Reference Implementation, is a scaled-down version of JAX-RPC specifically tailored to the J2ME platform.

The J2ME JAX-RPC subset allows Java developers to create portable web services clients that can easily exchange data between clients and back-end servers. Using XML and the Simple Object Access Protocol (SOAP), JAX-RPC allows developers to dispatch Remote Procedure Calls (RPC) to web services running on a different machine, a different network, or in a different language. Using RPC, developers can call methods on remote objects as easily as they can call them on local objects.

Creating portable web services makes it possible for any *client* (a device or a component within a device) to interact with any *service* (a set of specific functionality), regardless of how that service is implemented, and regardless of whether it is local or on the net. Web services are designed to be platform and language independent and can be implemented using many different technologies.

## Contents of the JAX-RPC Optional Package

The J2ME Web Services Reference Implementation JAX-RPC Optional Package contains three parts:

- JAX-RPC Subset APIs - the Java APIs used to develop a specific implementation of a web service. For more information, see [“The JAX-RPC API Subset.”](#)
- Remote Method Invocation (RMI) APIs - Java classes included with the J2ME Web Services Reference Implementation to satisfy dependencies of JAX-RPC on CLDC-based platforms. For more information, see [“The Remote Method Invocation API.”](#)
- A Java Web Services-specific Service Provider Interface (SPI) - enables stubs to be portable and compatible across varying implementations of the Java Web Services Specification. For more information, see [“The Service Provider Interface.”](#)

---

## The J2ME Client Platform

The J2ME platform is a set of standard Java APIs defined through the Java Community Process<sup>SM</sup> (JCP<sup>SM</sup>). It provides a flexible user interface, robust security, and built-in network protocols. The J2ME platform can be tailored for a specific class of device by matching it with a specific configuration and profile, and it can be extended for a specific market by adding additional optional packages.

The J2ME Web Services Reference Implementation supports:

- The Connected Limited Device Configuration (CLDC), v1.x. The CLDC configuration is designed for small devices with 16-bit or 32-bit CPUs and 128 KB to 512 KB of memory.
- The Mobile Information Device Profile (MIDP), v1.x and 2.x. The MIDP profile is specifically designed for cell phones and entry-level PDAs, and provides the user interface, network connectivity, local data storage, and application management needed by these devices.

---

**Note** – The J2ME Web Services Specification allows support for the Connected Device Configuration (CDC), as well as other profiles. However, these are not supported by the Reference Implementation.

---



With CLDC and MIDP, the J2ME platform provides a complete Java runtime environment for wireless, handheld, and mobile devices. Installing the J2ME Web Services JAXP and JAX-RPC optional packages further extends this Java runtime environment by adding additional XML processing capabilities to the J2ME platform.

## Difference Between J2SE and J2ME

The primary difference between the J2ME Web Services JAX-RPC optional package and the standard version of JAX-RPC, is that the J2ME version is tailored to run effectively in the reduced memory environment found in wireless and handheld devices. As a consequence, the J2ME JAX-RPC version contains only a subset of the classes and interfaces of JAX-RPC.

---

## Web Services Definition Language Files

To implement a web service in the J2ME Web Services Reference Implementation, one first needs to describe the service using the Web Services Definition Language (WSDL). WSDL is an industry standard language that allows a service to be defined as a web service. This can either be done by hand-writing WSDL directly, or by using a tool such as the Java2WSDL tool available in JAX-RPC. The Java2WSDL tool takes as input a standard Java interface and produces a WSDL file that corresponds to the Java interface.

Once a WSDL file exists for a web service, the service needs to be implemented and tied to the interface using the WSDL mapping. This process is known as generating a "tie" to the web service. The "tie" acts as a bridge between the generic description of the operations of the Web Service and its concrete Java implementation.

---

**Note** – The process of "tying" a service to an interface pertains only to web services implemented on the Java platform. Web services implemented on other platforms do not need to be tied.

---

For more information about the specific details of WSDL to Java mappings, see the *J2ME Web Services Specification, v1.0*.

---

## The JAX-RPC API Subset

The J2ME Web Services Reference Implementation JAX-RPC API subset contains the following four packages:

- `com.sun.j2mews.xml.rpc` - contains the runtime implementation classes.
- `javax.microedition.xml.rpc` - contains the runtime SPI interface classes.
- `javax.xml.namespace` - contains the Qualified Name (QName) class.
- `javax.xml.rpc` - contains the JAX-RPC API interface classes.

### `com.sun.j2mews.xml.rpc`

The `com.sun.j2mews.xml.rpc` package contains the following classes:

- `OperationImpl` - an implementation of the `javax.microedition.xml.rpc.Operation` class, corresponding to a `wsdl:operation` defined for a target service endpoint.
- `SOAPDecoder` - an implementation that decodes SOAP messages.
- `SOAPEncoder` - an implementation that encodes SOAP messages.

### `javax.microedition.xml.rpc`

The `javax.microedition.xml.rpc` package contains a single interface, `FaultDetailHandler`, which is implemented by stubs that handle custom faults.

This package also provides the following classes:

- `ComplexType` - provides a special Type instance used to represent an `xsd:complextype` defined in a Web Service's WSDL definition.
- `Element` - provides a special Object used to represent an `xsd:element` defined in a Web Service's WSDL definition.
- `Operation` - corresponds to a `wsdl:operation` defined for a target service endpoint.
- `Type` - provides a type safe enumeration of allowable types that are used to identify simple types defined in a Web Service's WSDL definition.

This package also provides a single exception, `FaultDetailException`, which returns service specific exception detail values, and an associated QName, to a Stub instance.

## javax.xml.namespace

The `javax.xml.namespace` package contains a single class, `QName`, which represents a qualified name as defined in the XML specifications: XML Schema Part2: Datatypes specification, Namespaces in XML, Namespaces in XML Errata.

## javax.xml.rpc

The `javax.xml.rpc` package contains a single interface, `Stub`, which is the interface for `javax.xml.rpc.Stub`, the common base interface for the stub classes.

This package also contains a single class, `NamespaceConstants`, used in JAX-RPC for namespace prefix.

This package also contains a single exception, `JAXRPCException`, which is thrown from the core JAX-RPC APIs to indicate an exception related to the JAX-RPC runtime mechanisms.

---

# The Remote Method Invocation API

The J2ME Web Services Reference Implementation provides one package that contains the RMI API. This is `java.rmi`, which provides classes needed by JAX-RPC to satisfy dependencies of JAX-RPC on CLDC platforms.

## java.rmi

The J2ME Web Services Reference Implementation `java.rmi` package contains the following single interface and three exceptions:

- `Remote` - this interface serves to identify interfaces whose methods may be invoked from a non-local virtual machine.
- `MarshalException` - this exception is thrown if a `java.io.IOException` occurs while marshalling the remote call header, arguments, or return value for a remote method call.
- `RemoteException` - this exception is the common superclass for a number of communication-related exceptions that may occur during the execution of a remote method call.
- `ServerException` - this exception is thrown as a result of a remote method invocation when a `RemoteException` is thrown while processing the invocation on the server, either while unmarshalling the arguments, executing the remote method itself, or marshalling the return value.

---

## The Service Provider Interface

The J2ME Web Services Reference Implementation provides only a single implementation of many that are possible, as other implementers writing to the J2ME Web Services Specification may produce a somewhat different implementation based upon the same guidelines and APIs.

In order to ensure that any implementation of the J2ME Web Services Specification will work with the J2ME Web Services Reference Implementation, an additional API is provided to ensure that all generated stubs are platform independent. This API is the Service Provider Interface (SPI), and it formally defines the interface between the J2ME Web Services Reference Implementation and the stub. By formally defining this interface, it ensures that any stub generated by any J2ME Web Services implementation will work with any other implementation that has correctly implemented the SPI.

The API defined by the J2ME Web Services Reference Implementation SPI is not found in the J2EE implementation of JAX-RPC.

## Sample JAX-RPC Web Service

---

The sample JAX-RPC web service (`EmployeeDB`, an employee database) is made up of three sample files. These are:

- `EmployeeDB.wsdl` - a sample WSDL file
- `EDBDMidlet2.java` - a sample Midlet file
- `config.xml` - a sample configuration file

The JAX-RPC web service and associated sample files are explained in detail in this chapter.

The complete code for the `EmployeeDB.wsdl` file can be found in [Appendix A](#); the `EDBDMidlet2.java` file in [Appendix B](#).

---

## The Web Services Client Model

The essence of RPC technology is the ability to perform functional calls to software running somewhere other than on a client. This could be on a different machine, a different platform, a different network, or in a different language. The goal of RPC is to make remote calls as easy for a developer as making local calls to local functions or objects.

A typical RPC scenario contains both a *service* and a *client*, where the service is a remote function or object, and the client is a local component contacted by a developer's application, in order to interact with the remote service. Once a web service has been deployed and is running, it is characterized by the following:

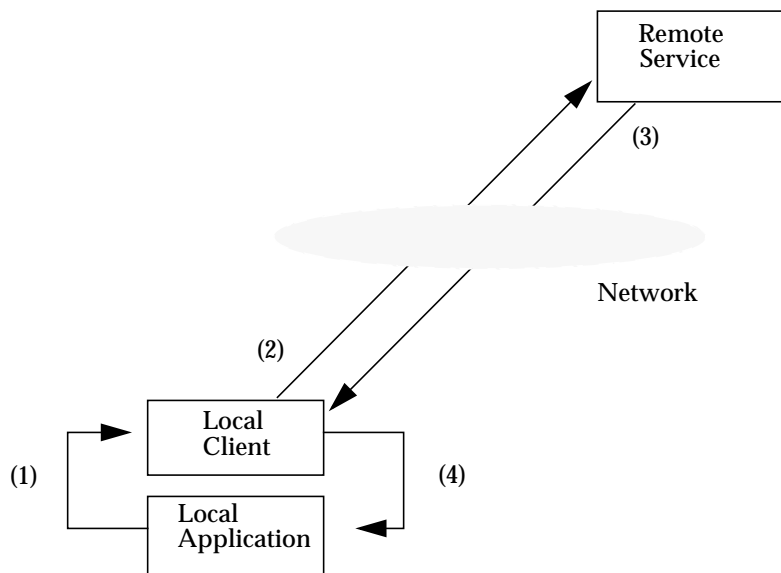
- An HTTP connection can be made to the service endpoint defined in the web service's WSDL definition file
- SOAP messages can be sent to the HTTP connection and SOAP messages are received in response

The following is a typical RPC interaction scenario:

1. A local application makes a procedure call to a local client component.
2. The local client component sets up a communication channel with the remote service and relays the procedure call to it.
3. The remote service accepts the procedure call, performs some processing, and returns a response to the local client via the communication channel.
4. The local client returns the response to the local application. The transaction appears to the local application as if it had occurred entirely on the client.

This transaction scenario is illustrated in [FIGURE 1](#).

**FIGURE 1** Typical RPC Scenario



Web services are designed to be platform and language independent. The goal is for any client to be able to interact with any web service, no matter how the client or the service is implemented. This interaction is made possible in two ways:

- The service developer describes the web service using a web services description language (WSDL) file. This defines the service endpoint needed by the client to make calls to the service.
- The client developer uses the web service WSDL file as input to the client stub generator, which outputs the Java code needed by a client application to make calls to the remote service.

Since the J2ME Web Services Reference Implementation is aimed at implementing JAX-RPC technology specifically on mobile and handheld client devices, the details of how to implement a Remote Service on the network (server) side of the equation are not discussed here.

---

**Note** – This document provides a sample WSDL file, `EmployeeDB.wsdl`, to illustrate an example client implementation. In an enterprise development context, the client developer must obtain, in advance, the WSDL file for the remote web service that is to be contacted.

---

---

## The `config.xml` Configuration File

Once you have acquired the WSDL file for the web service you want to communicate with, you need to create a configuration file that will be used by the Stub Generator to generate the stub (Java source files and compiled classes) needed to access the web service.

The following is the sample `config.xml` file to be used with the sample `EmployeeDB.wsdl` file.

### CODE EXAMPLE 1    Sample `config.xml` File

---

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration
  xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">
  <wsdl name="EmployeeDBService"
        location="EmployeeDB.wsdl"
        packageName="com.sun.xml.rpc.xml.EmployeeDB">
  </wsdl>
</configuration>
```

---

The `config.xml` file is a simple XML file you write yourself that defines the following four items:

- the XML namespace (`java.sun.com/xml/ns/jax-rpc/ri/config`)
- the name of the web service (`EmployeeDBService`)
- the location (name) of the WSDL file being used as input to the Stub Generator (`EmployeeDB.wsdl`)
- the package that will contain the Java classes and compiled files output by the Stub Generator (`com.sun.xml.rpc.EmployeeDB`)

---

## The Stub Generator

The Stub Generator (or stub compiler) is a JAX-RPC tool provided by the J2ME Web Services Reference Implementation. It is used to produce a *stub*, or client-side proxy, that can be called by an application to place calls to a remote web service. The stub generator takes a WSDL file as input and outputs the necessary Java code needed to place calls to the remote service.

The purpose of the Stub Generator is to create the necessary SOAP messaging structures, network connections, and underlying RPC coding needed to place remote calls over the web. This saves the developer from having to do this rigorous underlying code work.

## Generating a J2ME Web Services Stub

Generating a J2ME Web Services Stub using the Stub Generator (`wscompile.sh`) is a simple process. Follow these steps:

1. **Place the web service WSDL file** (`EmployeeDB.wsdl`) **and the** `config.xml` **file in the same directory as the Stub Generator.**

The default location for the Stub Generator is `<your_install_dir>/bin`.

2. **Type the following:**

```
% wscompile.sh -keep -gen:client -f:wsdl -verbose config.xml
```

This generates the stub output files, described in more detail below.

---

**Note** – The Stub Generator syntax used by `wscompile.sh` will be familiar to JAX-RPC developers. It has been replicated to keep things familiar.

---

## The `wscompile.sh` Options

The `wscompile.sh` command takes the form:

```
% wscompile.sh [options] configuration_file
```

The command line shown here (and in Step 2 above) uses the following options:

- `-keep`. Keeps the generated ( . java version) files. When the `-keep` option is not specified, the Stub Generator keeps the `.class` versions of the files and throws away the `.java` versions.
- `-gen:client`. Keeps client-side artifacts, such as Java class wrappers and stubs. (This is counterpart to the `-gen:server` option, which instructs the Stub Generator to keep server-side artifacts.)



- `-f:wsi`. Enables Web Services Interoperability (WSI) basic profile features (for document/literal and rpc/literal).
- `-verbose`. Prints command output to the screen. Turning off `-verbose` runs the command silently, without displaying output.
- `config.xml`. The name of the configuration file used as input.

For a complete list of `wscompile.sh` options and some usage examples, change directory to `<your_install_dir>/bin` and type:

```
% wscompile.sh
```

## Stub Generator Output

Running the command as shown above creates the following package directory:

```
<your_install_dir>/bin/com/sun/xml/rpc/EmployeeDB
```

Output from the Stub Generator consists of:

- a public class for each `complexType` defined in the web service's WSDL file. For example:
  - `AddGroups.class`
  - `EmployeeArrayType.class`
  - `EmployeeType.class`
  - `NameArrayType.class`
  - `NameType.class`
- a public interface for the `portType` and binding operations defined in the web services's WSDL file. For example: `EmployeeDBPort.class`.
- a stub file that implements the `portType` and binding class, for creating connections with the service endpoint and handling data streaming. For example: `EmployeeDBPort_Stub.class`.

## The Stub

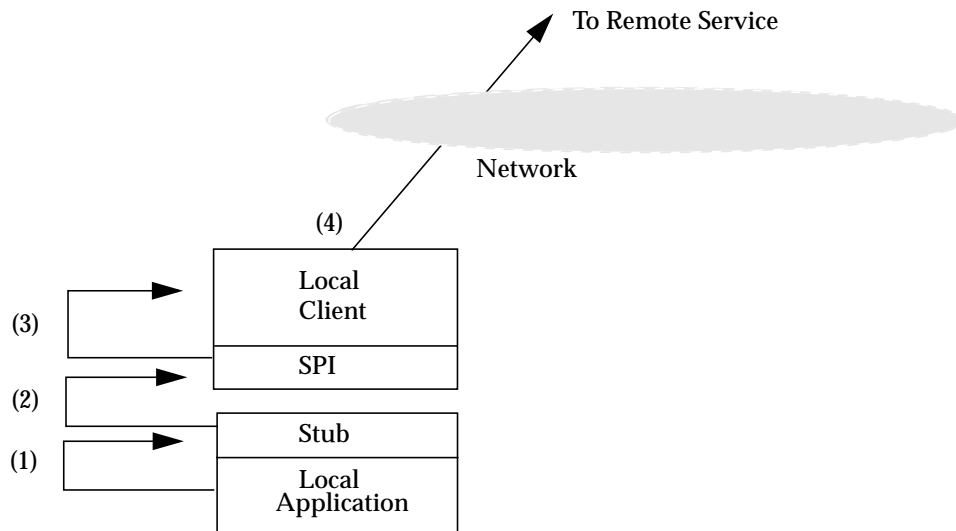
In order to ensure that any stub generated on any implementation of J2ME Web Services will work with any other implementation, the J2ME Web Services provides a Service Provider Interface (SPI) to ensure that transparency and compatibility exists in all generated stubs.

After a stub has been generated by Stub Generator (using the steps shown above), the generated stub can make calls to the implementation's SPI. There is nothing the application needs to do to ensure implementation independence.

The interaction between the local application, the stub, the Service Provider Interface, and the Java Web Services implementation takes place according to the following steps (as illustrated in [FIGURE 2](#)):

1. The local application makes calls to the stub.
2. The stub makes calls to the Service Provider Interface of the local client.
3. The SPI makes calls to the local client (the Java Web Services implementation).
4. The local client makes calls across the network to the remote service (as shown in [FIGURE 1](#)).

**FIGURE 2** The Stub and The Service Provider Interface



---

## The Web Service WSDL File

A web service's WSDL file is a portable XML file that defines important mappings needed to facilitate communication between a J2ME Web Services client application and a remote web service. It defines:

- the XML namespace and other necessary schemas, such as WSDL-to-SOAP bindings
- the complex types, elements, port types, messages, and operations that describe the functionality of the web service
- the web service address in the form of a URI

This list is just a summary. For a complete explanation of the contents, structure, and requirements of a WSDL file, see the World Wide Web Consortium (W3C) *Web Services Description Language 1.1 Specification*.

## WSDL File Mappings

The purpose of a WSDL file is to provide a set of standardized XML mappings that can be fed into a Stub Generator to produce a *stub*, as described in [“The Stub Generator.”](#) The following examples show some of the mappings defined in the sample WSDL file, `EmployeeDB.wsdl`. To see the complete example file, see [Appendix A](#).

To generate output based on the sample WSDL file:

1. Cut and paste the contents of [Appendix A](#) into a text file.
2. Follow the directions provided in the section [“Generating a J2ME Web Services Stub.”](#)

## Schema and Namespace Definition Statement

Like other XML files, a WSDL file opens with a schema and namespace definition statement. This statement defines the target namespace (`http://www.sun.com/EmployeeDB`) and schemas used in the file (XML, WSDL, SOAP, and an internal document schema).

### CODE EXAMPLE 2 Sample Schema and Namespace Definition

---

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions targetNamespace="http://www.sun.com/EmployeeDB"
xmlns="http://schemas.xmlsoap.org/wsdl/"
    name="EmployeeDB"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:tns="http://
www.sun.com/EmployeeDB"xmlns:xsd1="http://www.sun.com/
EmployeeDB.xsd">
```

---

## The type and complexType Definitions

The `complexType` definition is a specific item contained inside a more general type definition. They may be only one type block, but it can contain any number of `complexType` statements.

### CODE EXAMPLE 3 Sample complexType Definition

---

```
<type>
(.....)
<xsd:complexType name="NameArrayType">
  <xsd:sequence>
    <xsd:element name="Name" maxOccurs="unbounded"
      minOccurs="0" nillable="true" type="xsd1:NameType"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:element name="NameArray" type="xsd1:NameArrayType"/>
```

---

When the Stub Generator is run, it creates a public `NameArrayType` class. The sequence of `xsd:element` types shown here (`NameType` and `NameArrayType`) are converted by the Stub Generator into methods defined within the class.

This is equally true for the other `complexType` statements in the type definition (`NameType`, `EmployeeType`, `EmployeeArrayType`). Each `complexType` statement is converted by the Stub Generator into its own public class file with the `xsd:element` names converted into methods defined within the class.

## The portType Definition

The `portType` definition inside the sample WSDL file is used by the Stub Generator to create the `EmployeeDBPort` public interface file (`EmployeeDBPort.class`). This interface maps request operations to request methods which are then implemented in the stub file `EmployeeDBPort_Stub.class`.

The following example shows just one operation (`getEmployees`) of many that are mapped into the `EmployeeDBPort.class` file.

### CODE EXAMPLE 4 Sample portType Declaration

---

```
<portType name="EmployeeDBPort">
  <operation name="getEmployees">
    <input message="tns:getEmployeesReq"/>
    <output message="tns:getEmployeesRes"/>
  </operation>
```

---

This mapping produces the corresponding lines of Java code in the `EmployeeDBPort.class` file.

#### CODE EXAMPLE 5 Sample portType Java Code Output

---

```
public dom.sun.xml.rpc.EmployeeDB.EmployeeArrayType  
getEmployees(com.sun.xml.rpc.EmployeeDB.NameArrayType  
    employeesReq)  
throws java.rmi.RemoteException;
```

---

The `EmployeeDB.wsdl` file contains many other portType operations that get mapped by the Stub Generator. For a more complete understanding of these mappings, it is useful to compare the definitions in the WSDL file against the contents of the generated output.

## The binding Definition

The binding definition binds portType operations (`EmployeeDBPort`) to specific soap method calls which access endpoint properties, in the generated public class file `EmployeeDBPort_Stub.class`.

#### CODE EXAMPLE 6 Sample binding Definition

---

```
<binding name="EmployeeDBBinding" type="tns:EmployeeDBPort">  
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>  
    <operation name="getEmployees">  
        <soap:operation soapAction="http://www.sun.com/EmployeeDB/getEmployees"/>  
        <input name="getEmployeesReq">  
            <soap:body use="literal"/>  
        </input>  
        <output name="getEmployeesRes">  
            <soap:body use="literal"/>  
        </output>  
    </operation>
```

---

The public class file `EmployeeDBPort_Stub.class` is the stub file called by the sample MIDlet (`EDBDMidlet2.java`) to access the remote web service. For more information, see [“Packaging Stub and Data Classes with a MIDlet.”](#)

---

## Packaging Stub and Data Classes with a MIDlet

Once a stub has been generated, it becomes available to be called by a client application, such as the sample MIDlet, `EDBDMidlet2.java`. (For the complete code contents of the sample MIDlet, see [Appendix B](#).)

A client application developer writes calls to the stub in her application, and the stub manages the relationship to the implementation, which manages the cross-network RPC connections, SOAP message packaging, data transfer, and so on with the remote web service.

The client-application-to-web-service interaction takes place according to this sequence:

1. The J2ME client application makes programmatic calls to the stub.
2. The stub makes calls to the implementation via the formally defined SPI.
3. The J2ME Web Services implementation uses the information received from the stub via the SPI to open a network connection to the web service, perform the requested operation, and return the result to the stub via the SPI.
4. The stub receives the result and returns it to the J2ME client application as a result of the programmatic call.

## Making The Stub Available to An Application

To make a stub available to be called by an application, the developer can:

- Install the client application into the same directory as the stub.
- Bundle the stub with the application into a jar file and install the jar onto the appropriate platform.

## Programming Stub Calls into An Application

The sample MIDlet provides several examples of how programmatic calls to the stub can be written into J2ME client application code.

## Importing The Stub Package

Like any Java application, the sample MIDlet starts by importing the stub package along with other imported packages, as shown in the following example.

**CODE EXAMPLE 7** Importing the Stub Package

---

```
import javax.microedition.midlet.*;
(.....)
import com.sun.xml.rpc.xml.EmployeeDB.*;
```

---

## Defining the Stub and Creating a Stub Instance

The sample MIDlet first defines the stub using the previously generated public class, `EmployeeDBPort_Stub`, and then creates a new instance of the stub using the imported stub constructor, as shown in the following example.

**CODE EXAMPLE 8** Defining a Stub and Creating a Stub Instance

---

```
public class EDBDMidlet2 extends MIDlet
    implements CommandListener, Runnable
{
    (.....)
    EmployeeDBPort_Stub stub;
    (.....)
    stub = new EmployeeDBPort_Stub();
```

---

## Interacting with the Web Service Using the Stub

The sample `EDBDMidlet` sets up a program for interacting with an employee database web service from a handheld device. It creates the screen layout, establishes forms and queries, and then seeks over the network to establish a connection to the remote database web service. It provides a set of commands for interacting with the MIDlet, and uses calls to the stub to add, remove, promote, and demote employees in the remote database.

The following code sample makes calls to the stub to execute the methods `stub.addEmployee()`, `stub.removeEmployee()`, `stub.promoteEmployee()`, and `stub.demoteEmployee()`, methods that were defined when the stub was initially generated from the WSDL file.

For a look at the complete example, see [Appendix B, “EDBDMidlet2.java Sample File.”](#)

**CODE EXAMPLE 9** Making Calls to a Remote Web Service Using the Stub

---

```
public void run() {
    try {
        int sI = -1;
        boolean result = false;
        EmployeeType emp = null;
        switch (appState) {
            case 0:
                String fName = firstName.getString();
                String lName = lastName.getString();
                try {
                    int empid = stub.addEmployee(fName, lName);
                    (.....)
                } catch (Exception e) {
                    break;
                }
            case 1:
                sI = listScr.getSelectedIndex();
                emp = (EmployeeType)empList.elementAt(sI);
                try {
                    result = stub.removeEmployee(emp.getEmpID());
                    (.....)
                } catch (Exception e) {
                    break;
                }
            case 2:
                sI = listScr.getSelectedIndex();
                emp = (EmployeeType)empList.elementAt(sI);
                try {
                    result = stub.promoteEmployee(emp.getEmpID());
                    (.....)
                } catch (Exception e) {
                    break;
                }
            case 3:
                sI = listScr.getSelectedIndex();
                emp = (EmployeeType)empList.elementAt(sI);
                try {
                    result = stub.demoteEmployee(emp.getEmpID());
                    (.....)
                } catch (Exception e) {
                    break;
                }
        }
    }
}
```

---



## EmployeeDB.wsdl Sample File

This Appendix contains the complete text of the sample file `EmployeeDB.wsdl`, referred to in [Chapter 2, “Sample JAX-RPC Web Service.”](#) Portions of the file referred to in that discussion are highlighted here for your easy reference.

**CODE EXAMPLE 10** `EmployeeDB.wsdl` Sample File

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions targetNamespace="http://www.sun.com/EmployeeDB"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  name="EmployeeDB"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:tns="http://www.sun.com/EmployeeDB"
  xmlns:xsd1="http://www.sun.com/EmployeeDB.xsd">
  <types>
    <xsd:schema elementFormDefault="qualified"
      targetNamespace="http://www.sun.com/EmployeeDB.xsd"
      xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:xsd1="http://www.sun.com/EmployeeDB.xsd">
      <xsd:complexType name="NameType">
        <xsd:sequence>
          <xsd:element name="firstName" type="xsd:string"/>
          <xsd:element name="lastName" type="xsd:string"/>
        </xsd:sequence>
      </xsd:complexType>
      <xsd:element name="Name" type="xsd1:NameType"/>
    </xsd:complexType name="NameArrayType">
      <xsd:sequence>
        <xsd:element name="Name" maxOccurs="unbounded" minOccurs="0"
          nillable="true" type="xsd1:NameType"/>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="NameArray" type="xsd1:NameArrayType"/>
  </xsd:complexType name="EmployeeType">
    <xsd:sequence>
      <xsd:element name="empName" type="xsd1:NameType"/>
    </xsd:sequence>
  </xsd:complexType>
</definitions>
```

```

        <xsd:element name="empID" type="xsd:int"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="EmployeeArrayType">
    <xsd:sequence>
        <xsd:element name="Employee" maxOccurs="unbounded"
            minOccurs="0"
            nillable="true" type="xsd1:EmployeeType"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:element name="GetEmployees" type="xsd1:NameArrayType"/>
<xsd:element name="EmployeeArray" type="xsd1:EmployeeArrayType"/>
<xsd:element name="IsManager" type="xsd:int"/>
<xsd:element name="PromoteEmployee" type="xsd:int"/>
<xsd:element name="DemoteEmployee" type="xsd:int"/>
<xsd:element name="AddEmployee" type="xsd1:NameType"/>
<xsd:element name="EmployeeId" type="xsd:int"/>
<xsd:element name="RemoveEmployee" type="xsd:int"/>
<xsd:element name="Reset" type="xsd:boolean"/>
<xsd:element name="RetVal" type="xsd:boolean"/>
</xsd:schema>
</types>
<message name="getEmployeesReq">
    <part name="getEmployeesReqPart" element="xsd1:GetEmployees"/>
</message>
<message name="getEmployeesRes">
    <part name="getEmployeesResPart" element="xsd1:EmployeeArray"/>
</message>
<message name="isManagerReq">
    <part name="isManagerReqPart" element="xsd1:IsManager"/>
</message>
<message name="isManagerRes">
    <part name="isManagerResPart" element="xsd1:RetVal"/>
</message>
<message name="promoteEmployeeReq">
    <part name="promoteEmployeeReqPart"
        element="xsd1:PromoteEmployee"/>
</message>
<message name="promoteEmployeeRes">
    <part name="promoteEmployeeResPart" element="xsd1:RetVal"/>
</message>
<message name="demoteEmployeeReq">
    <part name="demoteEmployeeReqPart"
        element="xsd1:DemoteEmployee"/>
</message>
<message name="demoteEmployeeRes">
    <part name="addEmployeeResPart" element="xsd1:RetVal"/>
</message>
<message name="addEmployeeReq">
    <part name="addEmployeeReqPart" element="xsd1:AddEmployee"/>
</message>
<message name="addEmployeeRes">

```

```

        <part name="addEmployeeResPart" element="xsd1:EmployeeId"/>
    </message>
    <message name="removeEmployeeReq">
        <part name="removeEmployeeReqPart"
            element="xsd1:RemoveEmployee"/>
    </message>
    <message name="removeEmployeeRes">
        <part name="removeEmployeeResPart" element="xsd1:RetVal"/>
    </message>
    <message name="resetReq">
        <part name="removeEmployeeReqPart" element="xsd1:Reset"/>
    </message>
    <message name="resetRes">
        <part name="removeEmployeeResPart" element="xsd1:RetVal"/>
    </message>
    <portType name="EmployeeDBPort">
        <operation name="getEmployees">
            <input message="tns:getEmployeesReq"/>
            <output message="tns:getEmployeesRes"/>
        </operation>
        <operation name="isManager">
            <input message="tns:isManagerReq"/>
            <output message="tns:isManagerRes"/>
        </operation>
        <operation name="promoteEmployee">
            <input message="tns:promoteEmployeeReq"/>
            <output message="tns:promoteEmployeeRes"/>
        </operation>
        <operation name="demoteEmployee">
            <input message="tns:demoteEmployeeReq"/>
            <output message="tns:demoteEmployeeRes"/>
        </operation>
        <operation name="addEmployee">
            <input message="tns:addEmployeeReq"/>
            <output message="tns:addEmployeeRes"/>
        </operation>
        <operation name="removeEmployee">
            <input message="tns:removeEmployeeReq"/>
            <output message="tns:removeEmployeeRes"/>
        </operation>
        <operation name="reset">
            <input message="tns:resetReq"/>
            <output message="tns:resetRes"/>
        </operation>
    </portType>
    <binding name="EmployeeDBBinding" type="tns:EmployeeDBPort">
        <soap:binding style="document" transport="
http://schemas.xmlsoap.org/soap/http/>
    <operation name="getEmployees">
        <soap:operation soapAction="http://www.sun.com/EmployeeDB/getEmployees/">
        <input name="getEmployeesReq">
        <soap:body use="literal"/>

```

```

    </input>
    <output name="getEmployeesRes">
      <soap:body use="literal"/>
    </output>
  </operation>
<operation name="isManager">
  <soap:operation soapAction="http://www.sun.com/EmployeeDB/
isManager"/>
  <input name="isManagerReq">
    <soap:body use="literal"/>
  </input>
  <output name="isManagerRes">
    <soap:body use="literal"/>
  </output>
</operation>
<operation name="promoteEmployee">
  <soap:operation soapAction="http://www.sun.com/EmployeeDB/
promoteEmployee"/>
  <input name="promoteEmployeeReq">
    <soap:body use="literal"/>
  </input>
  <output name="promoteEmployeeRes">
    <soap:body use="literal"/>
  </output>
</operation>
<operation name="demoteEmployee">
  <soap:operation soapAction="http://www.sun.com/EmployeeDB/
demoteEmployee"/>
  <input name="demoteEmployeeReq">
    <soap:body use="literal"/>
  </input>
  <output name="demoteEmployeeRes">
    <soap:body use="literal"/>
  </output>
</operation>
<operation name="addEmployee">
  <soap:operation soapAction="http://www.sun.com/EmployeeDB/
addEmployee"/>
  <input name="addEmployeeReq">
    <soap:body use="literal"/>
  </input>
  <output name="addEmployeeRes">
    <soap:body use="literal"/>
  </output>
</operation>
<operation name="removeEmployee">
  <soap:operation soapAction="http://www.sun.com/EmployeeDB/
removeEmployee"/>
  <input name="removeEmployeeReq">
    <soap:body use="literal"/>
  </input>
  <output name="removeEmployeeRes">

```

```

        <soap:body use="literal"/>
    </output>
</operation>
<operation name="reset">
    <soap:operation soapAction="http://www.sun.com/EmployeeDB/
reset"/>
    <input name="resetReq">
        <soap:body use="literal"/>
    </input>
    <output name="resetRes">
        <soap:body use="literal"/>
    </output>
</operation>
</binding>
<service name="EmployeeDatabase">
    <port binding="tns:EmployeeDBBinding" name="EmployeeDBPort">
        <soap:address location="http://localhost:8000/ccx/EmployeeDB"/>
    </port>
</service>
</definitions>

```

---



## EDBDMidlet2.java Sample File

---

This Appendix contains the complete text of the sample file `EDBDMidlet2.java`, referred to in [Chapter 2](#).” Portions of the file referred to in that discussion are highlighted here for your easy reference.

### CODE EXAMPLE 11 Sample EDBDMidlet2.java File

---

```
package com.sun.jldemo.jsr172;
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.microedition.rms.*;
import java.util.Vector;
import java.util.Enumeration;
import com.sun.xml.rpc.demo.xml.EmployeeDB.*;
/**
 * A JavaOne 2003 Demo Midlet for JSR 172
 */
public class EDBDemoMidlet2 extends MIDlet
    implements CommandListener, Runnable
{
    Command exitC;
    Command settingsC;
    Command backC;
    Command applyC;
    Command addC;
    Command removeC;
    Command promoteC;
    Command demoteC;
    Command goC;
    Command aboutC;
    Form homeScr;
    Form settingsScr;
    Form addScr;
    Form confirmScr;
    Form aboutScr;
    Form loadingScr;
    List listScr;
```

```

TextBox errorScr;
TextField stdEndpoint;
TextField firstName;
    TextField lastName;
Gauge progressG;
Display display;    // The display for this MIDlet
boolean resetDB;
int appState;        // One of four values:
                    // 0 = Add, 1 = Remove, 2 = Promote, 3 = Demote
boolean settings;
Vector empList;
Thread network;
NameType[] nullList;
    EmployeeDBPort_Stub stub;
public EDBDemoMidlet2() {
    display = Display.getDisplay(this);
    nullList = new NameType[0];
errorScr = new TextBox("Error:", null, 500, TextField.ANY);
    errorScr.setCommandListener(this);
homeScr = new Form("Welcome");
    homeScr.setCommandListener(this);
try {
    Image hI = Image.createImage(
        this.getClass().getResourceAsStream("172splash.png"));
    homeScr.append(
        new ImageItem(null, hI, Item.LAYOUT_CENTER, null));
    } catch (Throwable t) {
        homeScr.append("Welcome to\n J2ME Web Services");
    }
settingsScr = new Form("Settings");
    settingsScr.setCommandListener(this);
stdEndpoint =
    new TextField("Standard Endpoint", "http://", 255,
TextField.ANY);
    settingsScr.append(stdEndpoint);
addScr = new Form("Add Employee");
    addScr.setCommandListener(this);
    firstName = new TextField("First Name", null, 50,
        TextField.ANY |
TextField.INITIAL_CAPS_WORD);
    lastName = new TextField("Last Name", null, 50,
        TextField.ANY |
TextField.INITIAL_CAPS_WORD);
    addScr.append(firstName);
    addScr.append(lastName);
confirmScr = new Form("Are you sure?");
    confirmScr.setCommandListener(this);

```



```

aboutScr = new Form("Credits");
    aboutScr.setCommandListener(this);
    StringItem si = new StringItem("JSR 172", null);
    si.setLayout(Item.LAYOUT_CENTER | Item.LAYOUT_NEWLINE_AFTER);
    aboutScr.append(si);
    si = new StringItem(null, "Jon Ellis    Mark Young");
    si.setLayout(Item.LAYOUT_CENTER);
    aboutScr.append(si);
    si = new StringItem(null, "Cliff Draper");
    si.setLayout(Item.LAYOUT_CENTER | Item.LAYOUT_NEWLINE_AFTER);
    aboutScr.append(si);
loadingScr = new Form("Please Wait");
    loadingScr.setCommandListener(this);
    progressG = new Gauge("Contacting Network...",
                          false,
                          Gauge.INDEFINITE,
                          Gauge.CONTINUOUS_RUNNING);
    progressG.setLayout(Item.LAYOUT_NEWLINE_BEFORE);
    loadingScr.append(progressG);
listScr = new List("Employee Database", Choice.IMPLICIT);
listScr.setCommandListener(this);
exitC = new Command("Exit", Command.EXIT, 1);
settingsC = new Command("Settings", Command.SCREEN, 1);
backC = new Command("Back", Command.BACK, 1);
applyC = new Command("Apply", Command.SCREEN, 1);
addC = new Command("Add", Command.SCREEN, 2);
removeC = new Command("Remove", Command.SCREEN, 3);
promoteC = new Command("Promote", Command.SCREEN, 4);
demoteC = new Command("Demote", Command.SCREEN, 5);
goC = new Command("Go", Command.SCREEN, 2);
aboutC = new Command("About", Command.SCREEN, 1);
// init the empList
empList = null;
// create stub instance
    stub = new EmployeeDBPort_Stub();
settings = loadSettings();
}
boolean loadSettings() {
    try {
        RecordStore rs = RecordStore.openRecordStore("SETTINGS",
false);
        stdEndpoint.setString(new String(rs.getRecord(1)));
        rs.closeRecordStore();
        return setSettings();
    } catch (RecordStoreException rse) {
        rse.printStackTrace();
    }
}
return false;
}

```

```

boolean setSettings() {
    stub.setProperty(
        javax.xml.rpc.Stub.ENDPOINT_ADDRESS_PROPERTY,
        stdEndpoint.getString());
    // NOTE: may possibly due some validation checking
    // of the values. If we return false, the 'settings'
    // screen will remain
    settings = true;
    return settings;
}
void saveSettings() {
    try {
        RecordStore rs = RecordStore.openRecordStore("SETTINGS",
            true);
        byte b[] = stdEndpoint.getString().getBytes();
        if (rs.getNextRecordID() > 1) {
            rs.setRecord(1, b, 0, b.length);
        } else {
            rs.addRecord(b, 0, b.length);
        }
        rs.closeRecordStore();
    } catch (RecordStoreException rse) {
        rse.printStackTrace();
    }
}
void showErrorScreen(String text) {
    clearCommands(errorScr);
    errorScr.setString(text);
    errorScr.addCommand(backC);
    display.setCurrent(errorScr);
}
void showHomeScreen() {
    clearCommands(homeScr);
    homeScr.addCommand(exitC);
    if (settings) {
        homeScr.addCommand(goC);
    }
    homeScr.addCommand(settingsC);
    homeScr.addCommand(aboutC);
    display.setCurrent(homeScr);
}
void showSettingsScreen() {
    settingsScr.setTitle("Settings");
    clearCommands(settingsScr);
    settingsScr.addCommand(backC);
    settingsScr.addCommand(applyC);
    display.setCurrent(settingsScr);
}

```

```

void showListScreen() {
    clearCommands(listScr);
    listScr.addCommand(backC);
    listScr.addCommand(demoteC);
    listScr.addCommand(addC);
    listScr.addCommand(removeC);
    listScr.addCommand(promoteC);
    display.setCurrent(listScr);
}
void showAboutScr() {
    clearCommands(aboutScr);
    aboutScr.addCommand(backC);
    display.setCurrent(aboutScr);
}
void showAddScr() {
    clearCommands(addScr);
    addScr.addCommand(backC);
    addScr.addCommand(addC);
    display.setCurrent(addScr);
}
void showConfirmScr(int mode) {
    clearCommands(confirmScr);
    confirmScr.deleteAll();
    confirmScr.addCommand(backC);
    int s = listScr.getSelectedIndex();
    StringItem si = null;
    switch (mode) {
        // Remove Employee
        case 0:
            si = new StringItem(
                "Are you sure you want to remove the Employee:\n",
                "\n" + listScr.getString(s));
            confirmScr.append(si);
            confirmScr.addCommand(removeC);
            break;
        // Promote Employee
        case 1:
            si = new StringItem(
                "Are you sure you want to promote the Employee:\n",
                "\n" + listScr.getString(s));
            confirmScr.append(si);
            confirmScr.addCommand(promoteC);
            break;
        // Demote Employee
        case 2:
            si = new StringItem(
                "Are you sure you want to demote the Employee:\n",
                "\n" + listScr.getString(s));
            confirmScr.append(si);
            confirmScr.addCommand(demoteC);
            break;
    }
}

```

```

display.setCurrent(confirmScr);
}
void clearCommands(Displayable d) {
    d.removeCommand(exitC);
    d.removeCommand(settingsC);
    d.removeCommand(backC);
    d.removeCommand(applyC);
    d.removeCommand(addC);
    d.removeCommand(removeC);
    d.removeCommand(promoteC);
    d.removeCommand(demoteC);
    d.removeCommand(goC);
    d.removeCommand(aboutC);
}
void addEmp() {
    // Special case - Don't allow more than one network
    // operation
    if (network != null) {
        return;
    }
    display.setCurrent(loadingScr);
    appState = 0;
    network = new Thread(this);
    network.start();
}
void removeEmp() {
    // Special case - Don't allow more than one network
    // operation
    if (network != null) {
        return;
    }
    display.setCurrent(loadingScr);
    appState = 1;
    network = new Thread(this);
    network.start();
}
void promoteEmp() {
    // Special case - Don't allow more than one network
    // operation
    if (network != null) {
        return;
    }
    display.setCurrent(loadingScr);
    appState = 2;
    network = new Thread(this);
    network.start();
}
void demoteEmp() {
    // Special case - Don't allow more than one network
    // operation
    if (network != null) {
        return;
    }

```

```

    }
    display.setCurrent(loadingScr);
    appState = 3;
    network = new Thread(this);
    network.start();
}
// This method MUST only be called from the 'network' thread
void loadEmps() {
    if (resetDB) {
        // Make call to reset the database
        try {
            stub.reset(true);
        } catch (Throwable t) {
            showErrorScreen(t.getMessage());
            resetDB = false;
            return;
        }
        resetDB = false;
    }
    listScr.deleteAll();
    if (empList != null) {
        empList.removeAllElements();
    }
    EmployeeType emps[] = null;
    try {
        emps = stub.getEmployees(nullList);
        empList = new Vector(emps.length);
        for (int i = 0; i < emps.length; i++) {
            empList.addElement((Object)emps[i]);
        }
    } catch (Throwable t) {
        showErrorScreen(t.getMessage());
        return;
    }
    if (emps == null) {
        return;
    }
    // Loop through the array, adding a new element to
    // listScr for each Employee, in the form
    // "EmpID - LastName, FirstName"
    StringBuffer buf = new StringBuffer();
    EmployeeType emp;
    for (Enumeration e = empList.elements() ; e.hasMoreElements()
; ) {
        emp = (EmployeeType)e.nextElement();
        buf.setLength(0);
        if (emp.getEmpID() < 10) {
            buf.append("0");
        }
        buf.append(emp.getEmpID());
        buf.append(" - ");
        buf.append(emp.getEmpName().getFirstName());
    }
}

```

```

        buf.append(" ");
        buf.append(emp.getEmpName().getLastName());
        listScr.append(buf.toString(), null);
    }
    // If we want, we could make calls to determine if an
    // employee was a Manager, and represent that in the List
    // by a bold font, or extra field, or what not
    showListScreen();
}
public void startApp() {
    showHomeScreen();
}
public void pauseApp() {
}
public void destroyApp(boolean unconditional) {
    saveSettings();
}
public void commandAction(Command c, Displayable s) {
    if (c == exitC) {
        notifyDestroyed();
        return;
    }
    if (c == settingsC) {
        showSettingsScreen();
        return;
    }
    if (c == backC) {
        if (s == settingsScr || s == aboutScr
            || s == listScr || s == loadingScr
            || s == errorScr)
        {
            resetDB = true;
            showHomeScreen();
        }
        else if (s == confirmScr || s == addScr) {
            showListScreen();
        }
        return;
    }
    if (c == applyC) {
        saveSettings();
        if (setSettings()) {
            showHomeScreen();
        } else {
            settingsScr.setTitle("Settings: Retry");
        }
        return;
    }
    if (c == addC) {
        if (s == listScr) {
            showAddScr();
        }
    }
}

```

```

        else if (s == addScr) {
            addEmp();
        }
        return;
    }
    if (c == removeC) {
        if (s == listScr) {
            showConfirmScr(0);
        }
        else if (s == confirmScr) {
            removeEmp();
        }
        return;
    }
    if (c == promoteC) {
        if (s == listScr) {
            showConfirmScr(1);
        }
        else if (s == confirmScr) {
            promoteEmp();
        }
        return;
    }
    if (c == demoteC) {
        if (s == listScr) {
            showConfirmScr(2);
        }
        else if (s == confirmScr) {
            demoteEmp();
        }
        return;
    }
    if (c == goC) {
        // Special case - Don't allow more than one network
        // operation
        if (network != null) {
            return;
        }
        display.setCurrent(loadingScr);
        appState = -1;
        network = new Thread(this);
        network.start();
        return;
    }
    if (c == aboutC) {
        showAboutScr();
        return;
    }
}

// Contact the server and retrieve the list of employees,
// update the listScr, and show it when done
public void run() {

```

```

        try {
            int sI = -1;
            boolean result = false;
            EmployeeType emp = null;
            switch (appState) {
                case 0:
                    // Add Employee
                    String fName = firstName.getString();
                    String lName = lastName.getString();
                    try {
                        int empid = stub.addEmployee(fName, lName);
                        NameType name = new NameType();
                        name.setFirstName(fName);
                        name.setLastName(lName);
                        emp = new EmployeeType();
                        emp.setEmpID(empid);
                        emp.setEmpName(name);
                        empList.addElement((Object)emp);
                    } catch (java.rmi.RemoteException re) {
                        re.printStackTrace();
                    }
                    break;
                case 1:
                    // Remove Employee
                    sI = listScr.getSelectedIndex();
                    emp = (EmployeeType)empList.elementAt(sI);
                    try {
                        result = stub.removeEmployee(emp.getEmpID());
                        if (result)
                            empList.removeElementAt(sI);
                    } catch (java.rmi.RemoteException re) {
                        re.printStackTrace();
                    }
                    break;
                case 2:
                    // Promote Employee
                    sI = listScr.getSelectedIndex();
                    emp = (EmployeeType)empList.elementAt(sI);
                    try {
                        result = stub.promoteEmployee(emp.getEmpID());
                    } catch (java.rmi.RemoteException re) {
                        re.printStackTrace();
                    }
                    break;
                case 3:
                    // Demote Employee
                    sI = listScr.getSelectedIndex();
                    emp = (EmployeeType)empList.elementAt(sI);
                    try {
                        result = stub.demoteEmployee(emp.getEmpID());
                    } catch (java.rmi.RemoteException re) {
                        re.printStackTrace();
                    }

```



```

        }
    break;
    }
    appState = -1;
    loadEmps();
    network = null;
    return;
} catch (Throwable t) {
    System.err.println("here");
    t.printStackTrace();
    network = null;
    showErrorScreen(t.getMessage());
    return;
}
}
}

```



# Index

---

## A

API  
    JAX-RPC, 4  
    RMI, 5

## B

binding Definition, 15

## C

Client Model  
    Web Services, 7  
complexType Definition, 13  
config.xml File, 9  
Configuration, 9  
Connected Limited Device Configuration, 2

## J

J2ME  
    Client Platform, 2  
    Reference Implementation, 2  
JAX-RPC API, 4  
JAX-RPC Optional Package, 1

## M

Mobile Information Device Profile, 2

## N

Namespace Definition, 13

## O

Optional Package

JAX-RPC, 1

## P

portType Definition, 14

## R

Remote Method Invocation (RMI), 5  
RPC  
    Typical Scenario, 8

## S

Schema Definition, 13  
Service Provider Interface, 6, 12  
SPI, 6  
Stub, 11, 12  
    Available to Application, 16  
    Interacting with the Web Service, 17  
    Packaging with MIDlet, 16  
    Programming Calls to Application, 16  
Stub Generator, 10, 11  
Stub Instance, 17  
Stub Package, 17

## T

type Definition, 13

## W

Web Services  
    Client Model, 7  
Web Services Definition Language, 3  
wscompile.sh, 10  
WSDL, 3, 12, 13

