I n s i d e   W e b O b j e c t s

Web Services

For WebObjects 5.2

Preliminary

November 2002

Contents

Figures, Listings, and Tables

About This Book

This document studies example application projects to explain some of the concepts it addresses. This projects and other companion resources can be found in `/Developer/Documentation/WebObjects/Web_Services/projects`. Alternatively, you can download them from http://developer.apple.com/techpubs/webobjects.

The following list itemizes resources you can use to increase your Web services knowledge.

- The AmazonClient project in `/Developer/Examples/JavaWebObjects/AmazonClient` is an implementation of a client for Amazon.com Web services.
- *Building Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI* (Sams) gives great detail on the elements of Web-service development and deployment.
- *Architecting Web Services* (Apress) provides a high-level view of Web-service development.
- *Java & XML* (O'Reilly) introduces you to XML and processing XML documents using SAX (Simple API for XML).
- *Web Services Routing Protocol (WS-Routing)* (http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-routing.asp).
- Axis (http://www.apache.org/axis).
- *Simple Object Access Protocol (SOAP) 1.1* (http://www.w3.org/TR/2000/NOTE-SOAP-20000508).
- *Canonical XML Version 1.0* (http://www.w3.org/TR/2000/CR-xml-c14n-20001026).
- *Exclusive XML Canonicalization Version 1.0* (http://www.w3.org/TR/2001/WS-xml-exc-c14n-20011120).

- *Web Services Description Language (1.1)* (http://www.w3.org/TR/wsdl).

In this document, SOAP (Simple Object Access Protocol) refers to SOAP version 1.1.

C H A P T E R 2

Introduction to Web Services

This chapter introduces Web-service concepts as well as Web-service development in WebObjects.

What Are Web Services?

Web services provide an efficient way for applications to communicate with each other. Currently, many companies use electronic-data-interchange (EDI) systems to communicate with their business partners. EDI, however, requires the use of slow modems and dedicated phone lines. Also, a change in the structure of the data exchanged requires that the systems of all partners involved be updated. Web services, which are based on Simple Object Access Protocol (SOAP) messages that wrap Extensible Markup Language (XML) documents, provide a flexible infrastructure that leverages the ubiquitous HTTP (or HTTPS) over TCP/IP. This means that your organization probably has all the hardware and software infrastructure needed to deploy Web services already. In addition, thanks to XML's structure and flexibility, each partner can extract only the information it needs from a message, which gives participants a great deal of freedom.

But Web services provide more than an information-exchange system. When an application implements some of its functionality using Web services, it becomes more than the sum of its parts. For example, you can create a Web-service **operation** that uses a Web-service operation from another **provider** to give its **consumers** (also known as *service requestors*) information tailored to their needs. Web-service operations are akin to the methods of a Java class; a provider is an entity that publishes a Web service, while the entities that use the Web service are called consumers.

Current Web service technology allows an organization to easily integrate its systems, creating an enterprise-wide solution that leverages the work that is performed best by smaller groups within the enterprise. For example, the Payroll system is the one that should deal with an employee's compensation, while the Human Resources system is more appropriate for the management of vacation and sick-leave time. However, an Employee Information system should gather the information that both the Payroll and Human Resources systems contain, but should not duplicate it. The Employee Information system could display a window or Web page that an employee can view to analyze both her salary and accrued vacation time, without having to directly access the databases used by the other two systems. Payroll and vacation information would be available through Web service operations provided by separate applications, which are tailored for their particular objectives.

Web services can also be deployed over the Internet; however, you should ensure that sensitive information is not compromised. A SOAP message can hop through several computers across the Internet before reaching its destination, which exposes it to be viewed and modified by malicious entities. There are several standards and specifications that help you to protect the messages you send and to make sure that the ones you receive have not been compromised. See "Web Service Security" (page 29) for more information.

What Web services really provide is access to business logic. This business logic can be implemented in any language. Most companies implementing Web services for the first time will only add a Web-service front end to their existing applications. WebObjects makes this easy.

Web Service Discovery

The Web Service Description Language (WSDL) is a specification used to describe a Web service. This description allows an application to dynamically search for a service, find out what operations it provides, and invoke the operations it needs. A UDDI repository is a searchable directory of Web services that Web-service requestors can use to search for Web services. However, a Web service need not be published in a UDDI repository for an application to make use of it.

Axis

Axis is the third generation of Apache SOAP (an implementation of SOAP from the Apache Software Foundation). Axis is a SOAP engine as well as a code generator and WSDL processing tool. WebObjects uses Axis to both deploy and consume Web services.

The idea behind Axis is to serve as a bridge between your time-tested code and the world of Web services. By using Axis as its SOAP engine, WebObjects allows you to leverage the business logic you have already created and use it as the backbone of your Web services strategy.

Axis processes SOAP messages using a series of handlers, which are classes responsible for processing a message or part of a message in a certain way. In fact, you are free to add your own handlers to customize message processing. For more information on Axis, visit http://xml.apache.org/axis.

C H A P T E R 2

Introduction to Web Services

C H A P T E R 3

Web Service Essentials

You can think of Web services as distributed applications. Instead of instantiating an instance of a class and invoking its methods, a Web-service **consumer** locates a Web service and invokes the operations it provides. The Web-service **provider** (the application implementing the Web service) can be on the same Java virtual machine as the one using it, or it can be thousands of miles away. Furthermore, the applications may be written in different languages and running is disparate platforms. Because of this, Web service consumers as well as Web service providers need a way of transferring information that is language and platform independent. This is where SOAP lends a hand.

Web services are based on SOAP (Simple Object Access Protocol). It provides an infrastructure for the exchange of structured data in a distributed environment. SOAP itself is based on XML (Extensible Markup Language). XML is an SGML (Standard Generalized Markup Language)-based language that facilitates the structuring of data in documents. In addition, data elements in XML documents provide information about the data they contain through element names and attributes.

SOAP was created to facilitate the exchange of information by heterogeneous systems. XML provides it with structure through schemas and element scope through namespaces. SOAP is a transport-agnostic protocol: messages can be sent using HTTP, SMTP, and other protocols. For more on XML, including XML Schema and XML Namespaces, see *Extensible Markup Language (XML)* at http://www.w3.org/XML.

Web Services and SOAP

SOAP is the messaging mechanism that you use when you consume Web service operations or provide Web service operations to your clients.

All Web service communication is done through SOAP messages. These messages have an envelope, represented by the `SOAP-ENV:Envelope` element, and a body, enclosed by the `SOAP-ENV:Body` element, containing the message's content. In addition the `SOAP-ENV:Envelope` can contain a `SOAP-ENV:Header` element enclosing one or more header entries. The header mechanism is what provides SOAP with decentralized extensibility. This is how extensions such as Digital Signature and WS-Security (Web Services-Security) are implemented. For more information on signatures and security, see [“Web Services Security” chapter].

SOAP provides two ways for representing Web service-operation invokations: RPC (Remote Procedure Call) and messaging. RPC messaging provides a way of representing method invokations in SOAP messages. Because of this, however, the structure of the messages representing operation invocations is fairly rigid. SOAP messaging, on the other hand, provides greater flexibility; it allows a messages to contain arbitrary data elements. However, parsing such messages is more complicated. Because RCP messaging is the more widely used method of invoking Web services, it is the one used in WebObjects. You can use SOAP messaging if your application requires it, but you have to process the message to extract the appropriate data and must perform error checking.

Ingredients of a SOAP Message

As Figure 3-1 shows, a SOAP message, represented by the `SOAP-ENV:Envelope` element, contains a mandatory `SOAP-ENV:Body` element an optional `SOAP-ENV:Header` element.

Figure 3-1 Structure of a SOAP message

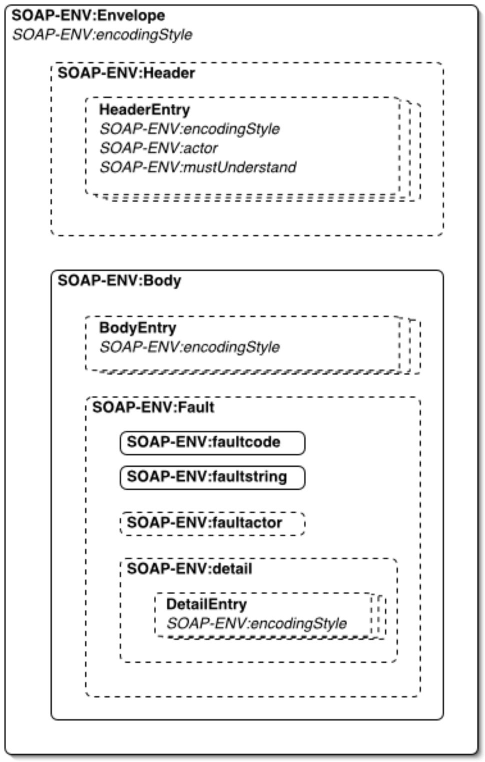


Table 3-1 describes the elements of a SOAP message.

Table 3-1 The elements of a SOAP message

Element	Parent	Use	Description
`SOAP-ENV:Envelope`	*None*	1	Root element of the message.
`SOAP-ENV:Header`	`SOAP-ENV:Envelope`	?	Encloses header entries.
Header entries	`SOAP-ENV:Header`	*	Heather entries provide additional information on the message's content. For example, digital signatures, authorization data, and so on.
`SOAP-ENV:Body`	`SOAP-ENV:Envelope`	1	Encloses the message's body entries.
Body entries	`SOAP-ENV:Body`	*	Body entries make up the content of the message. Their element names depend on the message's content.
`SOAP-ENV:Fault`	`SOAP-ENV:Body`	?	Body entry used to report a problem. When used, no other body entry can be present.
`SOAP-ENV:faultcode`	`SOAP-ENV:Fault`	1	Indicates the reason for the fault. Intended for application use.
`SOAP-ENV:faultstring`	`SOAP-ENV:Fault`	1	Human-readable version of the fault reason.
`SOAP-ENV:faultactor`	`SOAP-ENV:Fault`	?	Indicates which entity along the message path raised the fault.
`SOAP-ENV:detail`	`SOAP-ENV:Fault`	?	Encloses detail entries.
Detail entries	`SOAP-ENV:detail`	*	Contain application-specific information about the fault.

All the attributes that the SOAP envelope schema defines are global (they are not associated with a particular element). Also, each element in a SOAP message is free to use any attribute, regardless of where it's defined, either in SOAP's schema or another one, which is one of SOAP's extensibility features. This means that elements are free to use any number of attributes. Table 3-2 describes the attributes that the SOAP specification defines.

Table 3-2 Attributes defined in the SOAP specification

Attribute	Value	Description
`SOAP-ENV:actor`	A URI.	Specifies the entity that is to process the element. When absent the actor is the ultimate recipient of the message. This attribute is mainly used to assign header entries to specific entities.
`SOAP-ENV:mustUnderstand`	"`0`" or "`1`".	Indicates whether the element's actor must process the element. When set to `1` and if the actor is unable to process the element, the actor must respond with a `SOAP-ENV:Fault`.
`SOAP-ENV:encodingStyle`	A list of URIs.	Indicates the encoding style used for the element's content.

For more information on SOAP, see Simple Object Access Protocol at http://www.w3.org/TR.

Web Service Description

For a consumer to be able to use a Web service's operations, it must know what operations the Web service provides, the parameters they take, the type of the values they return, and so on. With intimate knowledge of the Web service, you can write a Web service client that takes full advantage of the service. However, the idea behind Web services is to provide a way for an application to dynamically find a Web service that satisfies its requirements and to learn how to use it. One of the building blocks that bring that vision closer to reality is WSDL (Web Services Description Language).

Like SOAP, WSDL is an XML-based language. A WSDL document tells a service requestor where a Web service is located and how to use it.

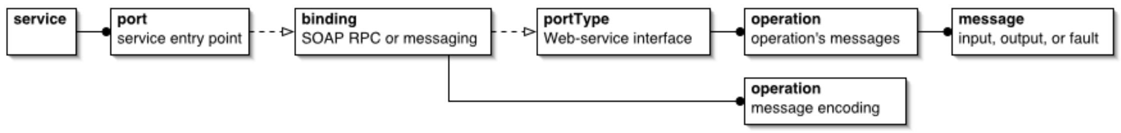
A WSDL document describes a Web service in two ways: an abstract description or interface and a concrete implementation. The interface section provides a high-level description of the operations the Web service provides and their parameter and return types. The implementation section binds each operation described in the interface section with its implementation (the methods that perform the work).

These are some of the XML elements that WSDL defines to describe a Web service:

- `portType`: This element provides the interface to the Web service. It describes each operation provided by the service as a set of input (from the consumer) and output (from the provider) messages that can be generated as a result of invoking the operation. Included in the list of possible messages are fault messages.
- `message`: This element describes a SOAP message. It lists the message's elements, which are referred to as *parts*, and their types.
- `types`: This elements list all the data types used as parameters or return types used in `message` elements. Essentially, it's an XML Schema definition.
- `binding`: This element specifies the transport used to send messages between the Web service consumers and its provider and implements a `portType`. It also defines the type of encoding used for each message.
- `port`: This element defines the URL that the Web service consumers use to access the Web service. It implements a `binding`.
- `service`: This element encloses one or more `port` elements.

Figure 3-2 shows the relationship between the major elements of a WSDL document. Missing from the figure are the `definitions` element, which is the root element of WSDL document and the `types` element.

Figure 3-2 Organization of a WSDL document



For the most part, you don't have to concern yourself with reading or writing WSDL documents. There are tools available that can both create Java classes from a WSDL document and generate a WSDL file from a set of Web-service implementation classes. For more information on WSDL, see *Web Services Description Language (WSDL)* at http://www.w3.org/TR.

C H A P T E R 4

Axis Basics

A SOAP processor aids both consumers of Web services and their providers to accomplish their task without having to worry about the intricacies of SOAP-message handling. As far as the consumer is concerned, it invokes an operation, in a similar way a remote procedure call is normally invoked. The Web service provider only needs to implement the logic required by the business problem it solves. The consumer's SOAP processor converts the method invocation into a SOAP message. This message is transmitted through a transport, such as HTTP or SMTP, to the service provider's SOAP processor, which parses the message into a method invocation. The provider then executes the appropriate logic and gives the result to the SOAP processor, which parses the information into a SOAP message. The message is transmitted through a transport to the consumer. It's SOAP processor parses the message and hands it out to the invoking code.

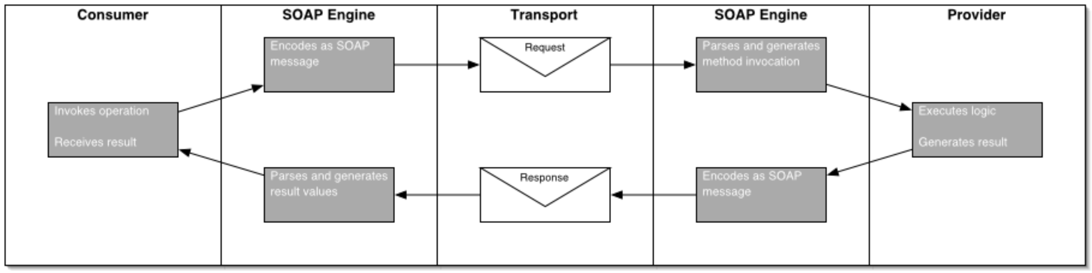
This chapter contains the following sections:

The Axis SOAP Engine

As mentioned in "Axis" (page 11), WebObjects uses the Axis framework to both serve and consume Web services. Axis is an interface between your business logic and the Web-services world.

The Axis Web service-processing model is shown in Figure 4-1.

Figure 4-1 The SOAP Message processing cycle



Axis implements a very extensible message-processing model. It uses **handlers** and **handler chains** to allow its functionality to be tailor to a wide variety of situations and requirements. A handler is an atomic component that acts on a specific part of a SOAP message; for example, a handler can be in charge of performing authentication on the message's sender before allowing it to be processed by the provider. A special handler, the *pivot handler* (another name for the service's provider), is in charge of executing the Web service's logic. It's called pivot handler because it is where the message's processing cycle changes from request processing to response processing.

A handler chain is a group of handlers that can be viewed as a unit. An important concept to grasp is that handlers and handler chains are not Web service specific. For example, you can develop handlers that process SOAP messages from transports other than HTTP or SMTP to increase security without having to change the Web service implementation. If you start now, you may be able to sell those handlers to others for a nice profit.

Handlers are simply Java classes that act on an `org.apache.axis.MessageContext` object. A MessageContext object contains several useful objects, but the most important are the `requestMessage` and the `responseMessage`. Handlers processing incoming messages normally access the `requestMessage` object, while those processing the outgoing message access the `responseMessage` object. However, Axis provides no restrictions; a handler can access and modify whatever it pleases. This is helpful if you need a handler to act both on incoming and outgoing messages.

Figure 4-2 shows the relationship between handlers and chains in Axis from the point of view of a Web service provider, while Figure 4-3 does the same from the perspective of a consumer.

Figure 4-2 Web service processing—provider view

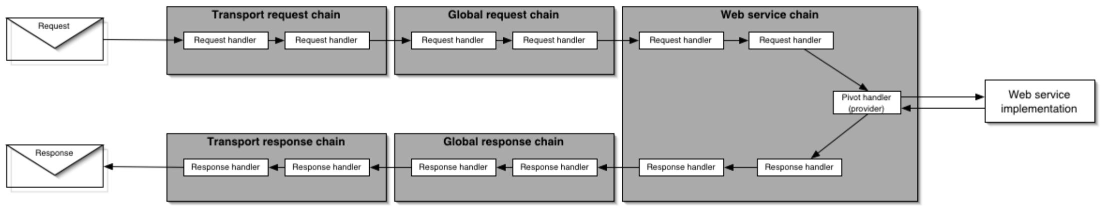
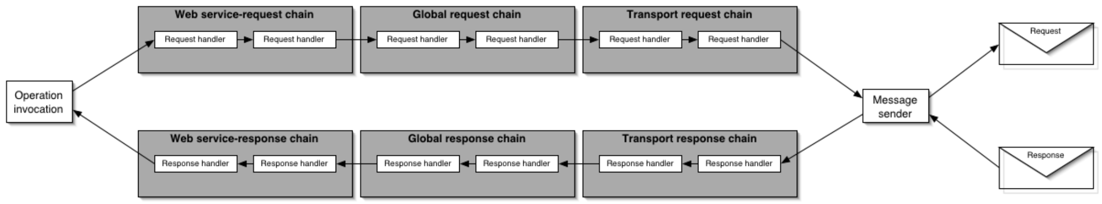


Figure 4-3 Web service processing—consumer view



Notice that there are three types of chains: transport, global, and Web service. A transport chain can deal with issues specific to the transport used to send and receive SOAP messages. A global chain is one that processes every SOAP message, regardless of the transport used and the target Web service. Finally, a Web service chain is one tailored for a specific Web service. For more information on handlers and chains, see Axis's documentation at http://xml.apache.org/axis.

Serialization and Deserialization of Objects

Complex classes require a custom serialization and deserialization strategy. WebObjects provides serializers and deserializers for [most of its classes]. If you have special classes that required a special serializer and deserializer, you have to create them.

Creating serializer and deserializer classes is a simple process, but requires knowledge of SAX (Simple API for XML). To learn how to process SAX callbacks in your serializers and deserializers, see *Java & XML* (O'Reilly).

C H A P T E R 5

Developing Web Services

You can publish as Web service operations the public methods of any class that contains a no-argument constructor. In addition, methods that need access WebObjects class must have an `org.apache.axis.MessageContext` as their first parameter. Also, methods that correspond to document-style operations, must return an `org.w3c.dom.Document`. You can find documentation for these classes at http://xml.apache.org/axis, and http://www.w3.org respectively.

Providing a Calculator Web Service

As a companion to this document, in `projects/Calculator`, you find the Calculator project. It's a simple WebObjects-application project used to build an application that serves a Web service called Calculator. The service provides four operations: `add`, `subtract`, `multiply`, and `divide`. The operations take two parameters of type `double` and return a value of type `double`. The `Calculator.java` class, the workhorse of the Calculator Web service, is listed in Listing 5-1.

Listing 5-1 `Calculator.java` class in Calculator project

```
public class Calculator extends Object {

    public static double add(double addend1, double addend2) {
        double sum = addend1 + addend2;
        return sum;
    }
    public static double subtract(double minuend, double subtrahend) {
        double difference = minuend - subtrahend;
        return difference;
    }

    public static double multiply(double multiplicand1, double multiplicand2) {
        double product = multiplicand1 * multiplicand2;
        return product;
    }

    public static double divide(double dividend, double divisor) {
        double quotient = dividend / divisor;
        return quotient;
    }
}
```

To provide a Web service based on `Calculator.java` the Application object registers Calculator.java as a Web service with the follwing method invocation:

```
WOWebServiceRegistrar.registerWebService(Calculator.class, true);
```

To become a Web service provider, build and run the Calculator application. To view the WSDL document for the Web service, point your Web browser to http://localhost:4210/WebObjects/Calculator.woa/ws/Calculator?wsdl.

Using the Calculator Web Service

The companion project Calculator_Client, located in `project/Calculator_Client`, contains the source files used to create the Calculator_Client application, which consumes the Calculator Web service described in "Providing a Calculator Web Service" (page 25). Its main class is `CalculatorClient.java`, listed in .

Listing 5-2 `CalculatorClient.java` class in Calculator_Client project

```
import java.net.*;
import java.util.Enumeration;
import com.webobjects.foundation.*;
import com.webobjects.webservices.client.*;

public class CalculatorClient extends Object {

    /**
     * Object through which the Web service's operations are invoked.
     */
    private WOWebServiceClient _serviceClient = null;

    /**
     * Address for the Web service's WSDL document.
     */
    private String _service_address = "http://localhost:4210/cgi-bin/WebObjects/
Calculator.woa/ws/Calculator?wsdl";

    /**
     */
    public CalculatorClient() {
        super();
    }

    /**
     * Obtains the Web service's operation names.
     * @return the Web service's operation names.
     */
    public NSArray operations() {
        NSArray operations =
(serviceClient().operationsDictionaryForService(serviceName())).allValues();
        NSMutableArray operation_names = new NSMutableArray();
        Enumeration operations_enumerator = operations.objectEnumerator();
        while (operations_enumerator.hasMoreElements()) {
            WOClientOperation operation =
(WOClientOperation)operations_enumerator.nextElement();
            operation_names.addObject((String)operation.name());
        }
        return operation_names;
    }

    /**
```

```
 * Invokes the Web service's operations.
 * @param operation      operation to invoke;
 * @param arguments      argument list;
 * @return               value returned by the operation.
 */
public Double invoke(String operation, Object[] arguments) {                    //1
    Object result = serviceClient().invoke(serviceName(), operation, arguments);
    return (Double)result;
}

/**
 * Obtains the Web service name.
 * Normally one WSDL file describes one Web service,
 * but it could describe one or more services.
 * @return Web service name.
 */
public String serviceName() {                                                   //2
    return (String)serviceClient().serviceNames().objectAtIndex(0);
}

/**
 * Obtains an agent through which service operations are invoked.
 * @return service agent.
 */
private WOWebServiceClient serviceClient() {
    if (_serviceClient == null) {
        _serviceClient = clientFromAddress(_service_address);
    }
    return _serviceClient;
}

/**
 * Obtains a Web service-client object through which
 * service operations can be invoked.
 * @return Web service-client object.
 */
private static WOWebServiceClient clientFromAddress(String address) {
    WOWebServiceClient service_client = null;

    // Create the Web service's URL.
    URL url;
```

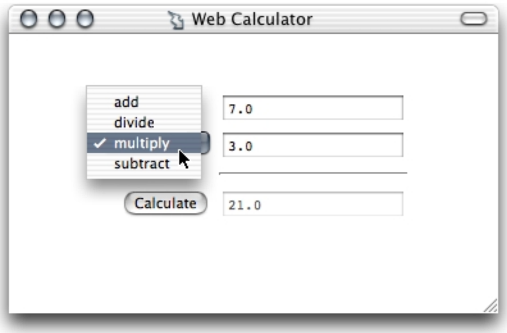
```
        try {
            url = new URL(address);
        }
        catch (MalformedURLException e) {
            url = null;
        }

        // Get a service-client object.
        service_client = new WOWebServiceClient(url);

        return service_client;
    }
}
```

The following list highlights some aspects of `CalculatorClient.java`.

1. The `invoke` method defines as parameters the operation name and its arguments. It uses the `invoke` method of WOWebServiceClient to invoke the Web service operation.
2. The `serviceName` method returns the name of the first Web service in the list of Web services defined by the WSDL document used to create the WOWebServiceClient object. Most WSDL documents define one Web service, but a WSDL document can define more than one Web service.

Build and run the Calculator_Client application. Your Web browser should show a page like the one shown in Figure 5-1. If your Web browser didn't launch, launch it and connect to http://localhost:5210/cgi-bin/WebObjects/Calculator_Client.woa.

Figure 5-1 A possible user interface to the Calculator Web service



Using Sessions

Using sessions during Web service consumption is simple. Each WOWebServiceClient object initializes a WOSession object when it's instantiated. However, you can change the session a client uses. For example, you can develop an application that provides several related Web services. A practical way to share information among the services is to store shared data in a session object.

The Security project, in `projects/Security`, showcases a simple Web service application that provides two Web services: LogIn and AccessData. The LogIn service accepts user data and stores in a session. To give the AccessData service access to the information recorded by LogIn, its session is set to the one used by LogIn.

The Security_Client project, in `projects/Security_Client`, implements a Web service client that consumes LogIn and AccessData. It sets user name and password properties through LogIn and retrieves them through AccessData. [listing] shows this process.

Listing 5-3 Security_Client project—`Application.java` file

```
import com.webobjects.appserver.*;
import com.webobjects.foundation.*;
import com.webobjects.webservices.client.*;

public class Application extends WOApplication {

    public static void main(String argv[]) {
        WOApplication.main(argv, Application.class);
    }

    public Application() {
        super();
        System.out.println("Welcome to " + this.name() + "!");

        // Create the service client used to consume
         // both LogInService and AccessDataService.
        SecurityClient securityClient = new SecurityClient();

        // Log in as Catherine with the password enirehtac.
        securityClient.logIn("Susana", "anasus");

        // Get session from LogInService.
        WOWebService.SessionInfo sessionInfo = securityClient.logInSessionInfo();

        // Set AccessDataService's session to the one obtained from LogInService.
        securityClient.setAccessDataSessionInfo(sessionInfo);

        // Get values of properties stored in session created by LogInService.
        String userName = securityClient.userName();
        String userPassword = securityClient.userPassword();

        // Print the properties' values.
        System.out.println();
        System.out.println("************************************************");
        System.out.println("User name from AccessDataService: " + userName);
        System.out.println("User password from AccessDataService: "  + userPassword);
        System.out.println("************************************************");
        System.out.println();
    }
}
```

Listing 5-4 shows the Service_Client class.

Listing 5-4 Security_Client project—`SecurityClient.java` file

```
import java.net.*;

import com.webobjects.appserver.*;
import com.webobjects.foundation.*;
import com.webobjects.webservices.client.*;

/**
 * Used to consume the LogIn and AccessData Web services.
 */
public class SecurityClient extends Object {

    private WOWebServiceClient _logInClient = null;
    private WOWebServiceClient _accessDataClient = null;

   private final String LogInServiceAddress = "http://localhost:4220/cgi-bin/WebObjects/
Security.woa/ws/LogIn?wsdl";
    private final String AccessDataServiceAddress = "http://localhost:4220/cgi-bin/
WebObjects/Security.woa/ws/AccessData?wsdl";

    private final String LogInService = "LogIn";
    private final String AccessDataService = "AccessData";


    public SecurityClient() {
        super();
    }

    /**
     * Invokes the setUserInfo operation of the LogIn Web service.
     */
    public void logIn(String name, String password) {
        Object[] arguments = { name, password };
        logInClient().invoke(LogInService, "setUserInfo", arguments);
    }
    /**
     * Invokes the userName operation of the AccessData Web service.
     * @return user name stored in shared session object.
     */
    public String userName() {
        Object result = accessDataClient().invoke(AccessDataService, "userName", null);
        return (String)result;
    }

    /**
     * Invokes the userPassword operation of the AccessData Web service.
     * @return user password stored in shared session object.
     */
    public String userPassword() {
        Object result = accessDataClient().invoke(AccessDataService, "userPassword",
null);
        return (String)result;
    }

    /**
     * Obtains a Web service client through which LogIn operations are invoked.
     * @return a Web service client for LogIn.
     */
    protected WOWebServiceClient logInClient() {
        if (_logInClient == null) {
            _logInClient = clientFromAddress(LogInServiceAddress);
        }
        return _logInClient;
    }

    /**
     * Obtains a Web service client through which AccessData operations are invoked.
     * @return a Web service client for AccessData.
     */
    protected WOWebServiceClient accessDataClient() {
        if (_accessDataClient == null) {
            _accessDataClient = clientFromAddress(AccessDataServiceAddress);
        }
        return _accessDataClient;
    }
    /**
     * Obtains session information from LogInService.
     * @return session information from LogInService.
     */
    public WOWebService.SessionInfo logInSessionInfo() {
        return logInClient().sessionInfoForServiceNamed(LogInService);
    }

    /**
     * Sets the session used by AccessDataService.
     */
    public void setAccessDataSessionInfo(WOWebService.SessionInfo sessionInfo) {
       accessDataClient().setSessionInfoForServiceNamed(sessionInfo, AccessDataService);
    }

    /**
     * Obtains a Web service client through which
     * service operations are invoked.
     * @return Web service client object.
     */
    private WOWebServiceClient clientFromAddress(String address) {
        WOWebServiceClient service_client = null;

        // Create the Web service's URL.
        URL url;
        try {
            url = new URL(address);
        }
        catch (MalformedURLException e) {
            url = null;
        }

        // Get a service-client object.
        service_client = new WOWebServiceClient(url);

        return service_client;
    }
}
```

Adding Web Service Support to Existing Projects

To add Web service–provider support to an existing project, you have to add the JavaWebServiceSupport framework to it. To add Web service-client support, you need to add the JavaWebServiceSupport and JavaWebServiceClient frameworks. The frameworks are located in `/System/Library/Frameworks` (`$NEXT_ROOT/Library/Frameworks on Windows`).

C H A P T E R 6

Developing Direct to Web Services Applications

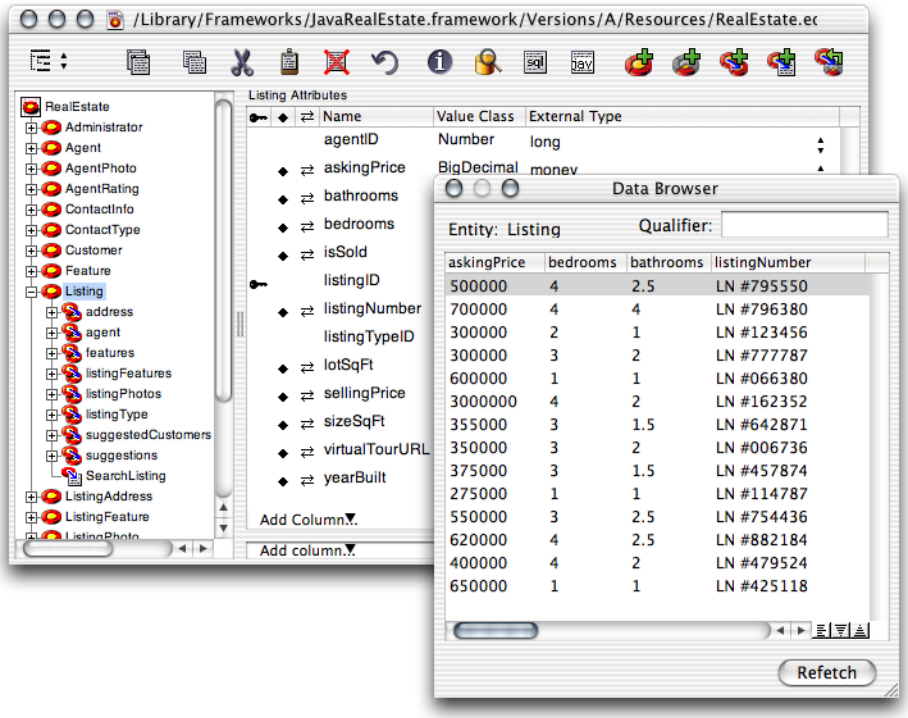
This chapter walks you through the creation of a Direct to Web Services application. Direct to Web Services allows you to rapidly develop Web service–based applications that provide access to a data store. As other WebObjects rapid-development approaches, Direct to Web Services is a data model–based and rule-based application development approach.

You create a project called `HousesForSale` that provides a Web service with two operations, one to find information on houses for sale and another to find real-estate agents. If you don't want to create the project by hand, you can find it in `projects/HomesForSale`.

The Data Model

The HousesForSale project includes JavaRealEstate framework located in `/Library/Frameworks`. The framework contains the RealEstate data model file. The data model defines several entities; you work with only two of them: Listing and ListingAddress. Figure 6-1 shows the Listing entity definition and data from its corresponding database table; Figure 6-2 does the same for the ListingAddress entity.

Figure 6-1 Listing entity defined in the RealEstate data model



/Library/Frameworks/JavaRealEstate.framework/Versions/A/Resources/RealEstate.eo

- RealEstate
  - Administrator
  - Agent
  - AgentPhoto
  - AgentRating
  - ContactInfo
  - ContactType
  - Customer
  - Feature
  - Listing
    - address
    - agent
    - features
    - listingFeatures
    - listingPhotos
    - listingType
    - suggestedCustomers
    - suggestions
    - SearchListing
  - ListingAddress
  - ListingFeature
  - ListingPhoto

Listing Attributes

🔑	◆	⇄	Name	Value Class	External Type
			agentID	Number	long
	◆	⇄	askingPrice	BigDecimal	money
	◆	⇄	bathrooms		
	◆	⇄	bedrooms		
	◆	⇄	isSold		
🔑			listingID		
	◆	⇄	listingNumber		
			listingTypeID		
	◆	⇄	lotSqFt		
	◆	⇄	sellingPrice		
	◆	⇄	sizeSqFt		
	◆	⇄	virtualTourURL		
	◆	⇄	yearBuilt		

Add Column

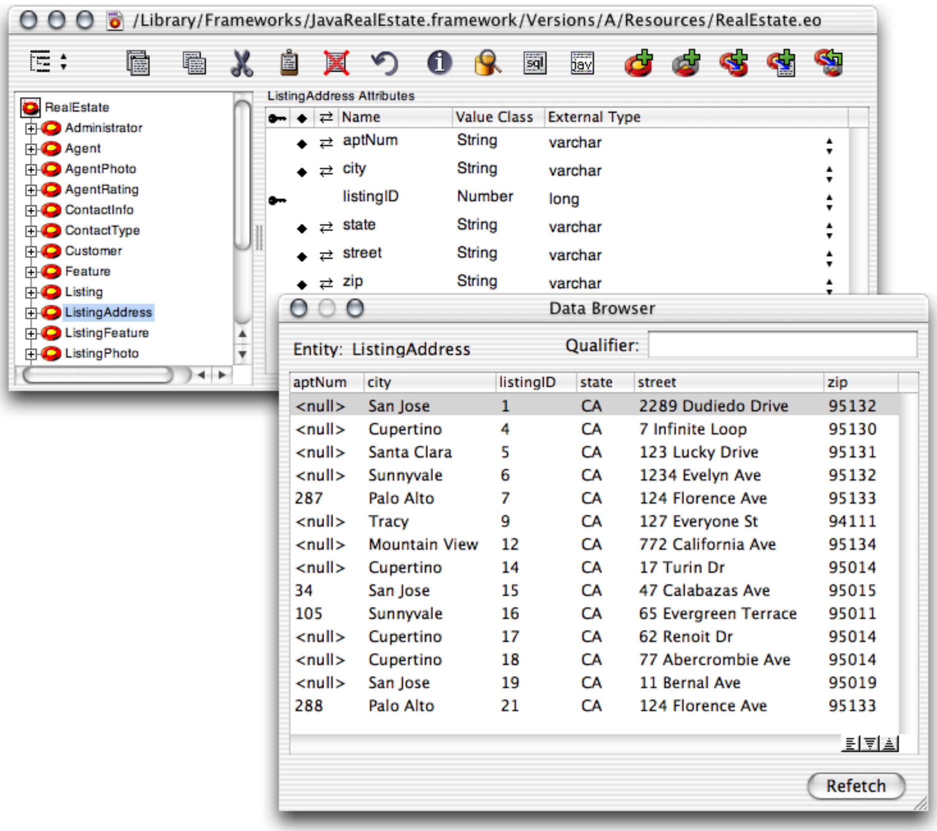
Add column

Data Browser

Entity: Listing Qualifier:

askingPrice	bedrooms	bathrooms	listingNumber
500000	4	2.5	LN #795550
700000	4	4	LN #796380
300000	2	1	LN #123456
300000	3	2	LN #777787
600000	1	1	LN #066380
3000000	4	2	LN #162352
355000	3	1.5	LN #642871
350000	3	2	LN #006736
375000	3	1.5	LN #457874
275000	1	1	LN #114787
550000	3	2.5	LN #754436
620000	4	2.5	LN #882184
400000	4	2	LN #479524
650000	1	1	LN #425118

Figure 6-2 ListingAddress entity defined in the RealEstate data model



/Library/Frameworks/JavaRealEstate.framework/Versions/A/Resources/RealEstate.eo

RealEstate
- Administrator
- Agent
- AgentPhoto
- AgentRating
- ContactInfo
- ContactType
- Customer
- Feature
- Listing
- ListingAddress
- ListingFeature
- ListingPhoto

ListingAddress Attributes

🔑	◆	⇄	Name	Value Class	External Type
	◆	⇄	aptNum	String	varchar
	◆	⇄	city	String	varchar
🔑			listingID	Number	long
	◆	⇄	state	String	varchar
	◆	⇄	street	String	varchar
	◆	⇄	zip	String	varchar

Data Browser

Entity: ListingAddress Qualifier:

aptNum	city	listingID	state	street	zip
<null>	San Jose	1	CA	2289 Dudiedo Drive	95132
<null>	Cupertino	4	CA	7 Infinite Loop	95130
<null>	Santa Clara	5	CA	123 Lucky Drive	95131
<null>	Sunnyvale	6	CA	1234 Evelyn Ave	95132
287	Palo Alto	7	CA	124 Florence Ave	95133
<null>	Tracy	9	CA	127 Everyone St	94111
<null>	Mountain View	12	CA	772 California Ave	95134
<null>	Cupertino	14	CA	17 Turin Dr	95014
34	San Jose	15	CA	47 Calabazas Ave	95015
105	Sunnyvale	16	CA	65 Evergreen Terrace	95011
<null>	Cupertino	17	CA	62 Renoit Dr	95014
<null>	Cupertino	18	CA	77 Abercrombie Ave	95014
<null>	San Jose	19	CA	11 Bernal Ave	95019
288	Palo Alto	21	CA	124 Florence Ave	95133

Creating the Project

Follow these steps to create a Direct to Web Services–based application:

1. Launch Project Builder, located in `/Developer/Applications`.

2. In the New Project pane of the Project Builder Assistant, select Direct to Web Services Application under WebObjects.
3. Name the project `HousesForSale`.
4. In the Choose EOAdaptors pane, make sure the JDBC adaptor is selected.
5. In the Choose Frameworks pane, add the JavaRealEstate framework located in `/Library/Frameworks`.
6. In the Build and Launch Project pane, deselect "Build and launch project now."
7. Edit the Properties file so that it looks like Listing 6-1.

Listing 6-1 Properties file of the HousesForSale project

```
WOAutoOpenInBrowser false
WOPort 5210
```

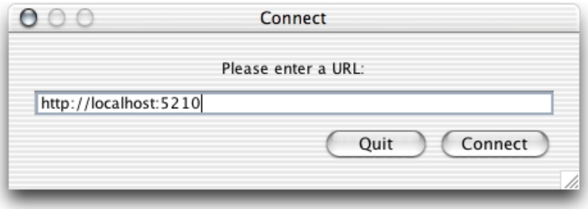
8. Build and run the application.

Web Services Assistant

To customize a Direct to Web Services application you use Web Services Assistant. It's located in `/Developer/Applications`.

After you launch Web Services Assistant, the Connect dialog appears (Figure 6-3). Enter `http://localhost:5210` in the text input field and click Connect.

Figure 6-3 Connect dialog of Web Services Assistant

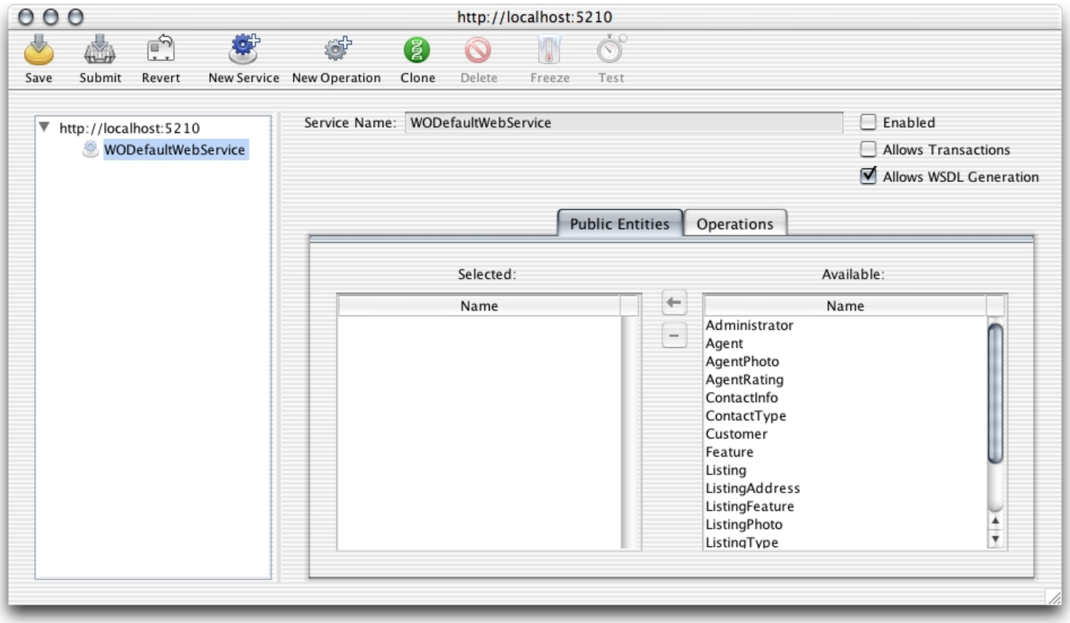

Connect

Please enter a URL:

http://localhost:5210

Figure 6-4 shows the Web Services Assistant main window.

Figure 6-4 The Web Services Assistant main window


http://localhost:5210

http://localhost:5210
WODefaultWebService

Service Name: WODefaultWebService

[ ] Enabled
[ ] Allows Transactions
[✓] Allows WSDL Generation

Selected:

Name

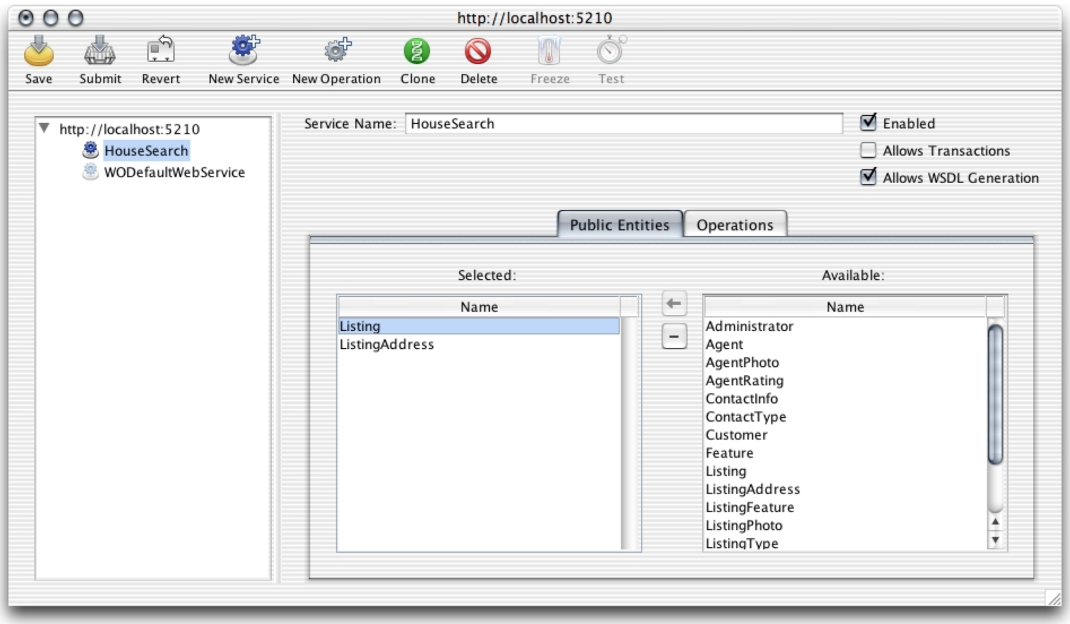
Available:

Name
Administrator
Agent
AgentPhoto
AgentRating
ContactInfo
ContactType
Customer
Feature
Listing
ListingAddress
ListingFeature
ListingPhoto
ListingType

Initially, your application contains one Web service named WODefaultWebService; the service is disabled. You should not enable this service. Instead, you should create a Web service and add operations to it. However, you can use the default Web service to create operations for your custom Web service.

Adding a Custom Web Service

1. In the Web Services Assistant main window, select `http://localhost:5210`.
2. Click the New Service button in the toolbar.
3. Enter `HouseSearch` in the Service Name text input field.
4. Make sure Enabled is selected.
5. Select Listing and ListingAddress in the Available list of the Public Entities pane and click the button with the left-pointing arrow.



http://localhost:5210
- HouseSearch
- WODefaultWebService

Service Name: HouseSearch

[✓] Enabled
[ ] Allows Transactions
[✓] Allows WSDL Generation

Selected:

Name
Listing
ListingAddress

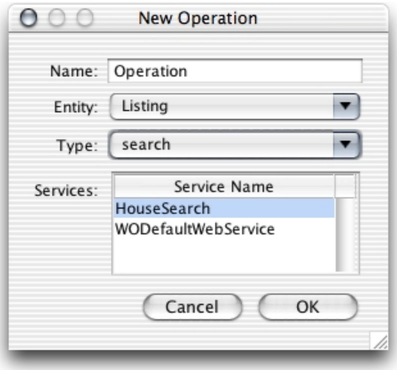
Available:

Name
Administrator
Agent
AgentPhoto
AgentRating
ContactInfo
ContactType
Customer
Feature
Listing
ListingAddress
ListingFeature
ListingPhoto
ListingType

Adding an Operation

1. Select HouseSearch under `http://localhost:5210`.
2. Click the New Operation button.
3. In the New Operation dialog (Figure 6-5), enter `findHouseByAskingPrice` in the Name text input field.
4. Choose Listing from the Entity pop-up menu.
5. Make sure Type is "search."
6. Make sure HouseSearch is selected in the Services list and click OK.

Figure 6-5 The New Operation dialog

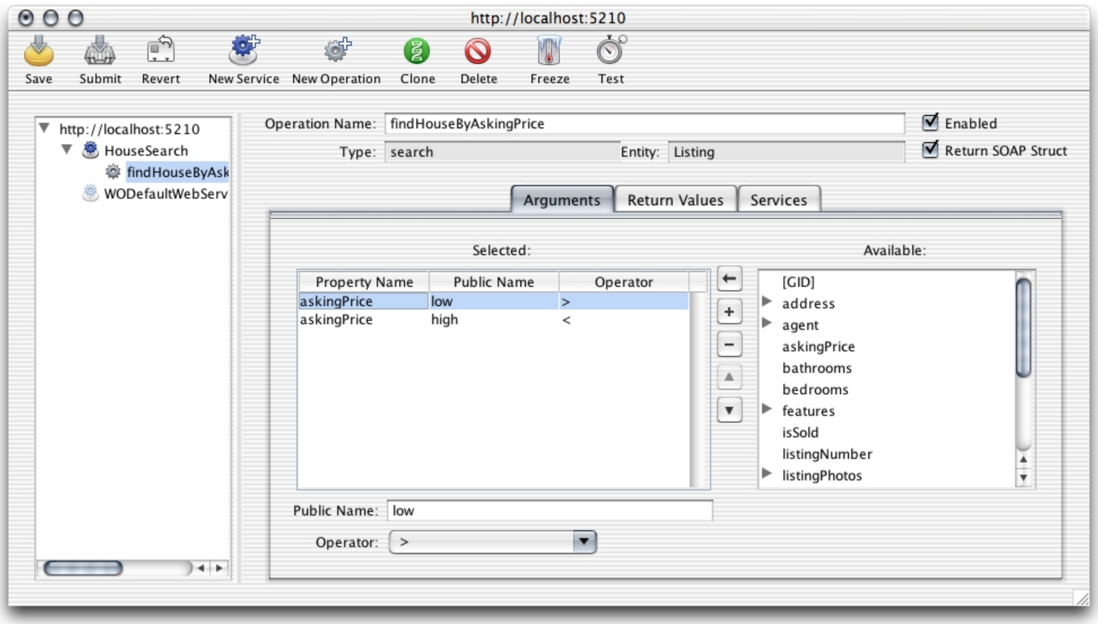


Name: Operation
Entity: Listing
Type: search
Services:

Service Name
HouseSearch
WODefaultWebService

7. In the main window (Figure 6-6), select `askingPrice` in the Available list in the Arguments pane and click the button with the left-pointing arrow twice.
8. Select the first row of the Selected list, enter `low` in the Public Name text input field, and choose ">" from the Operator pop-up menu.
9. Select the second row, enter `high` in the Public Name text field, and choose "<" from the Operator pop-up menu.

Figure 6-6 The findHouseByAskingPrice operation of the HouseSearch Web service



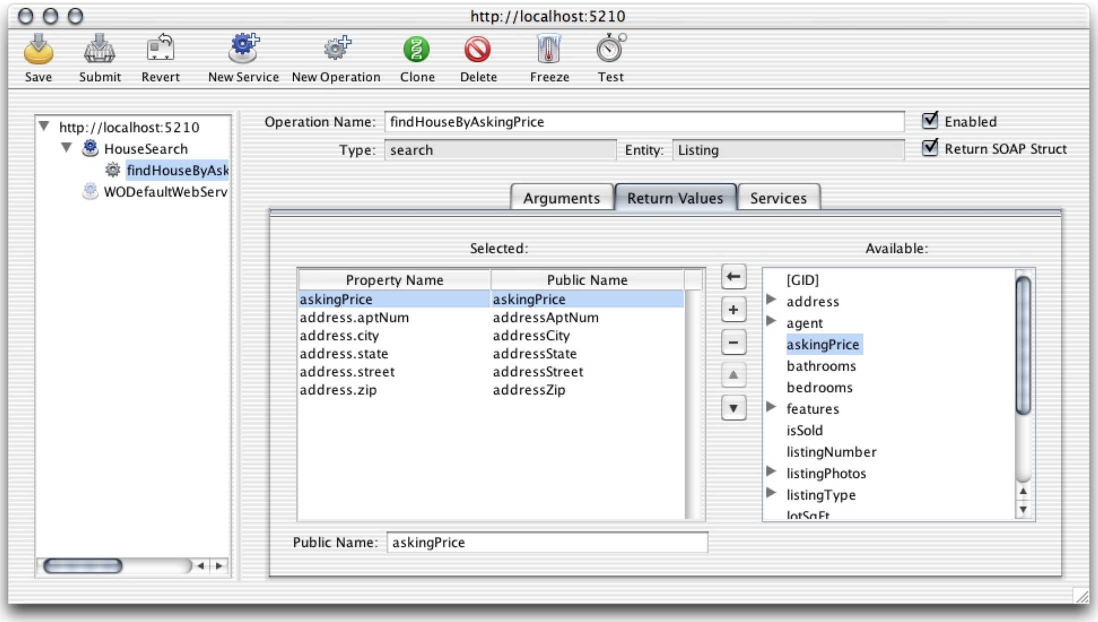
Operation Name: findHouseByAskingPrice
Type: search
Entity: Listing
☑ Enabled
☑ Return SOAP Struct

Property Name	Public Name	Operator
askingPrice	low	>
askingPrice	high	<

Public Name: low

Operator: >

10. In the Return Values pane, select `askingPrice` from the Available list and click the button with the left-pointing arrow. Repeat for `address.appNum`, `address.street`, `address.city`, `address.state`, and `address.zip`.



Operation Name: findHouseByAskingPrice
Type: search
Entity: Listing
☑ Enabled
☑ Return SOAP Struct

Property Name	Public Name
askingPrice	askingPrice
address.aptNum	addressAptNum
address.city	addressCity
address.state	addressState
address.street	addressStreet
address.zip	addressZip

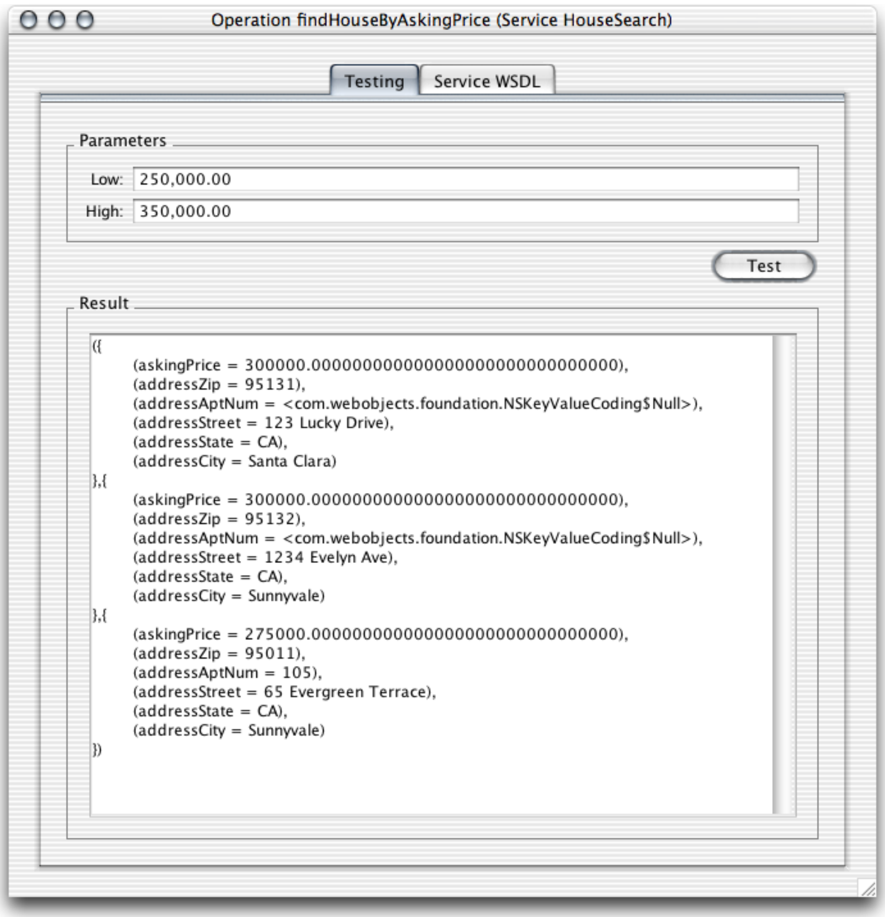
Public Name: askingPrice

Testing an Operation

1. Select `findHouseByAskingPrice` under HouseSearch under `http://localhost:5210`.
2. Select Return SOAP Struct.
3. Click the Test toolbar button.

   The test window ([figure]) has two panes: the Parameters pane and the Result pane. In the Parameters pane you enter the values for the operation's parameters. When you click Test, the Result pane shows the return values of the operation.
4. Enter `250000` in the Low text input field, `350000` in the High text field, and click Test.

Figure 6-7 The test window of the `findHouseByAskingPrice` operation



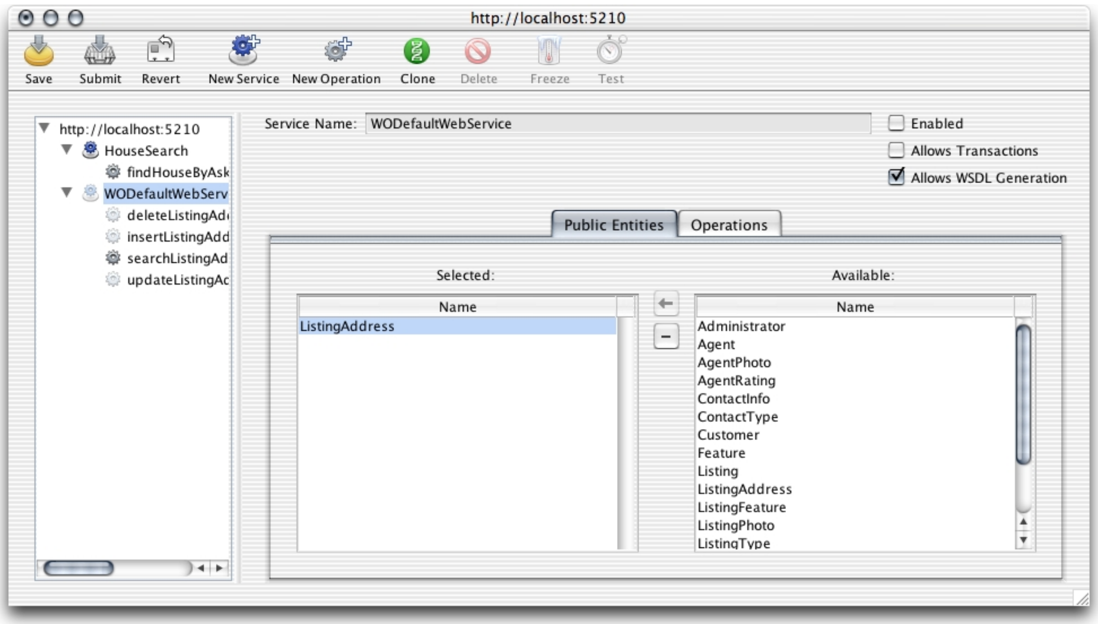
```
({
        (askingPrice = 300000.000000000000000000000000000000),
        (addressZip = 95131),
        (addressAptNum = <com.webobjects.foundation.NSKeyValueCoding$Null>),
        (addressStreet = 123 Lucky Drive),
        (addressState = CA),
        (addressCity = Santa Clara)
},{
        (askingPrice = 300000.000000000000000000000000000000),
        (addressZip = 95132),
        (addressAptNum = <com.webobjects.foundation.NSKeyValueCoding$Null>),
        (addressStreet = 1234 Evelyn Ave),
        (addressState = CA),
        (addressCity = Sunnyvale)
},{
        (askingPrice = 275000.000000000000000000000000000000),
        (addressZip = 95011),
        (addressAptNum = 105),
        (addressStreet = 65 Evergreen Terrace),
        (addressState = CA),
        (addressCity = Sunnyvale)
})
```

Using WODefaultWebService Operations

1. In the Web Service Assisntat main window, select WODefaultWebService under `http://localhost:5210`.

2. In the Public Entities pane, select ListingAddress in the Available list and click the button with the left-pointing arrow.

   Web Services Assistant adds four operations to WODefaultWebService: `deleteListingAddress`, `insertListingAddress`, `searchListingAddress`, and `updateListingAddress`. However, only the `searchListingAddress` operation is enabled.



http://localhost:5210
- HouseSearch
  - findHouseByAsk
- WODefaultWebServ
  - deleteListingAd
  - insertListingAdd
  - searchListingAd
  - updateListingAc

Service Name: WODefaultWebService

☐ Enabled
☐ Allows Transactions
☑ Allows WSDL Generation

Selected:

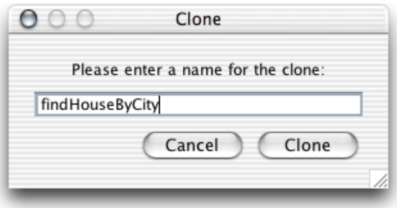
Name
ListingAddress

Available:

Name
Administrator
Agent
AgentPhoto
AgentRating
ContactInfo
ContactType
Customer
Feature
Listing
ListingAddress
ListingFeature
ListingPhoto
ListingType

3. Select `searchListingAddress` under WODefaultWebService.

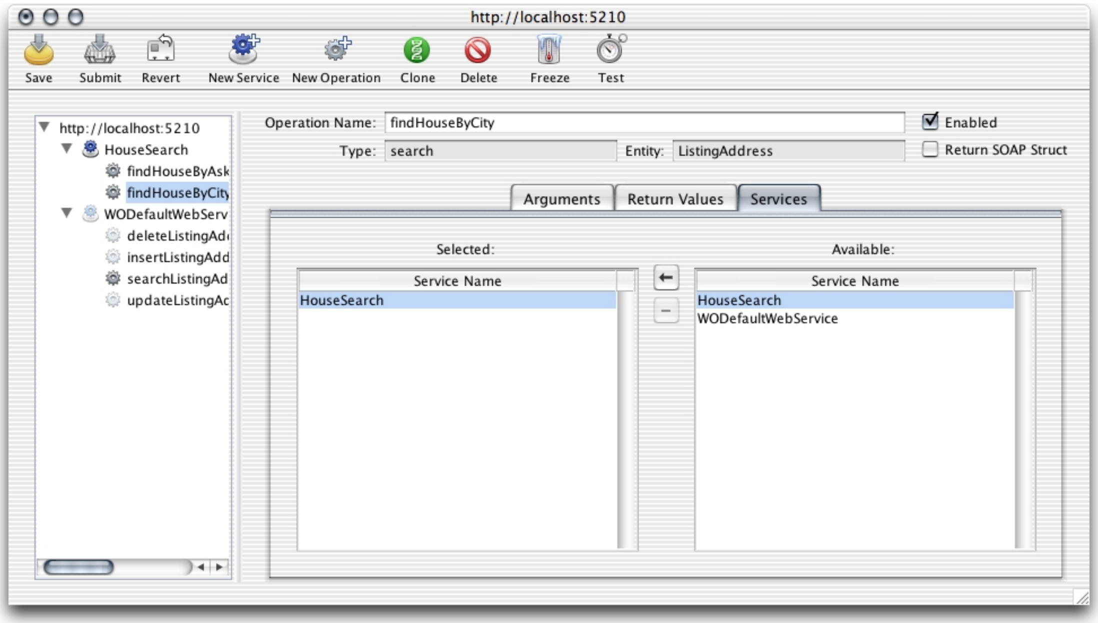
4. Click the Clone toolbar button, enter `findHouseByCity` in the text input field of the Clone dialog, and click Clone.



Please enter a name for the clone:

findHouseByCity

5. Select `findHouseByCity` under WODefaultWebService.
6. In the Services pane, select HouseSearch in the Available list and click the button wiht the left-pointing arrow.
7. In the Selected pane, select WODefaultWebService and click the button with the minus sign. The operation is now part of the HouseSearch Web service.

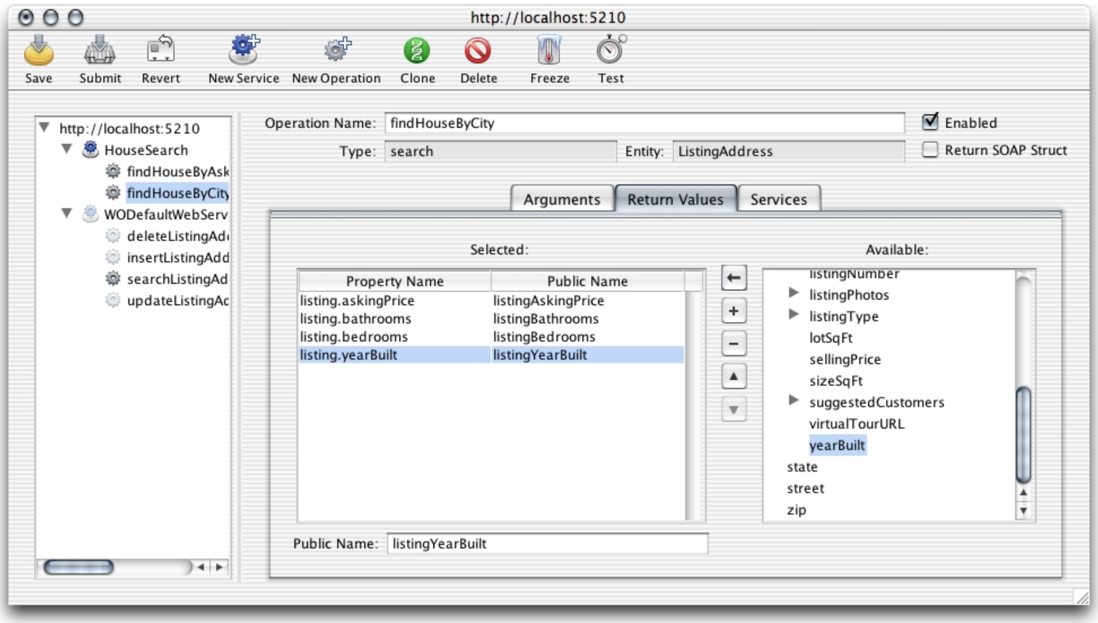


Operation Name: findHouseByCity
Enabled [✓]
Type: search
Entity: ListingAddress
Return SOAP Struct [ ]

Selected: Service Name
HouseSearch

Available: Service Name
HouseSearch
WODefaultWebService

8. Select findHouseByCity under HouseSearch and display the Arguments pane.
9. Select the `aptNum` property in the Selected list and click the button with the minus sign. Repeat for `street` and `zip`.
10. In the Return Values pane, remove all properties from the Selected list.
11. Select `listing.askingPrice` in the Available list and click the button with the left-pointing arrow. Repeat for `listing.bathrooms`, `listing.bedrooms`, and `listing.yearBuilt`.



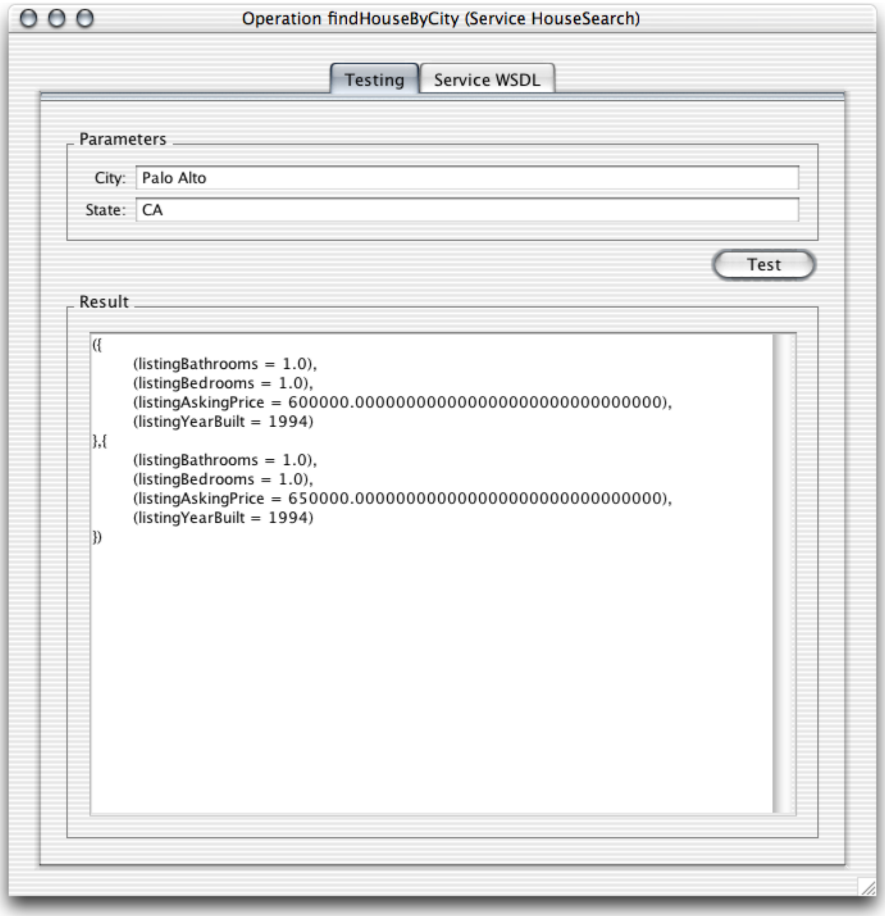
Operation Name: findHouseByCity [✓ Enabled]

Type: search Entity: ListingAddress [ ] Return SOAP Struct

Property Name	Public Name
listing.askingPrice	listingAskingPrice
listing.bathrooms	listingBathrooms
listing.bedrooms	listingBedrooms
listing.yearBuilt	listingYearBuilt

Public Name: listingYearBuilt

12. Select Return SOAP Struct and test the operation.

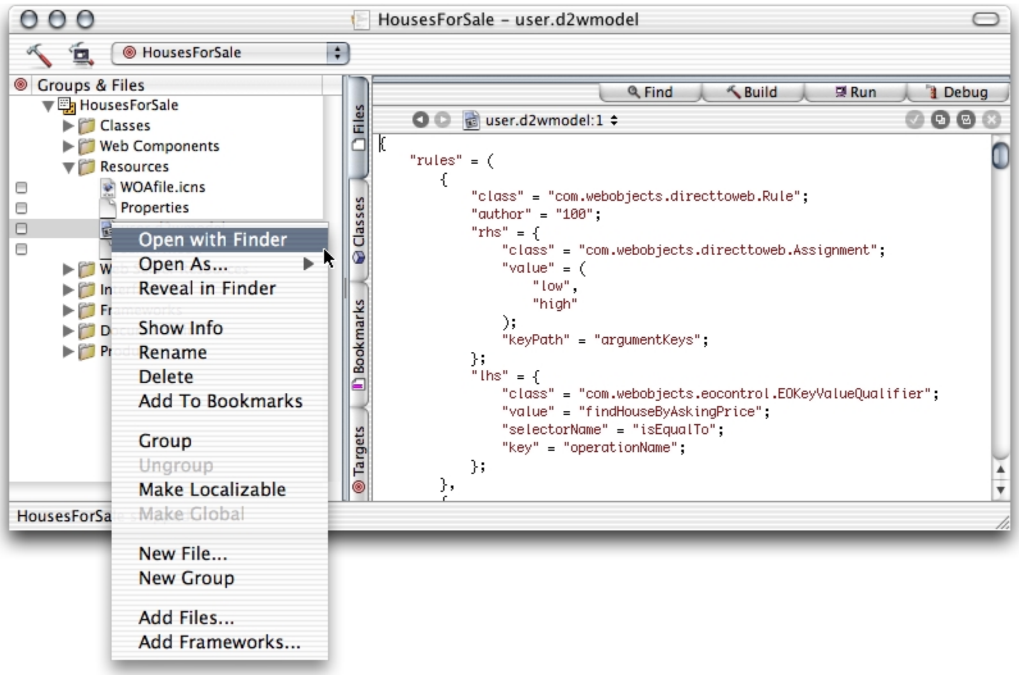


```
({
        (listingBathrooms = 1.0),
        (listingBedrooms = 1.0),
        (listingAskingPrice = 600000.000000000000000000000000000000),
        (listingYearBuilt = 1994)
},{
        (listingBathrooms = 1.0),
        (listingBedrooms = 1.0),
        (listingAskingPrice = 650000.000000000000000000000000000000),
        (listingYearBuilt = 1994)
})
```

Observing SOAP Messages Using TCPMonitor

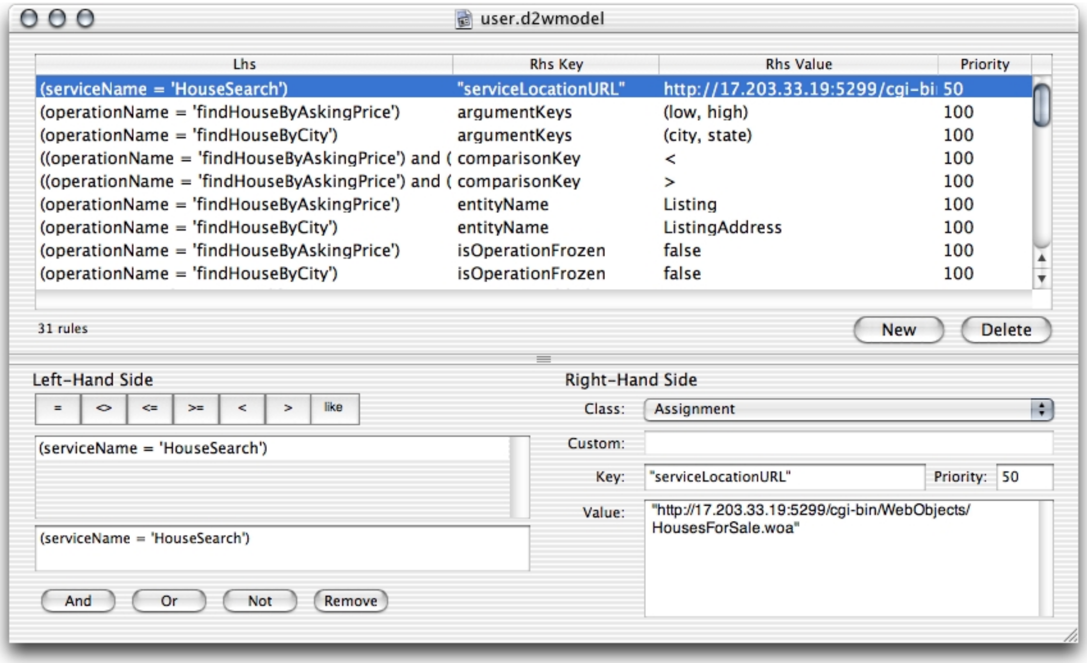
1. If you haven't saved the Web service's configuration in Web Service Assistant, do so now.

2. Open the project's `user.d2wmodel` file in Rule Editor by Control-clicking the `user.d2wmodel` file under the Resources group in the HouseSearch Project Builder main window and choosing Open with Finder.



```
{
    "rules" = (
        {
            "class" = "com.webobjects.directtoweb.Rule";
            "author" = "100";
            "rhs" = {
                "class" = "com.webobjects.directtoweb.Assignment";
                "value" = (
                    "low",
                    "high"
                );
                "keyPath" = "argumentKeys";
            };
            "lhs" = {
                "class" = "com.webobjects.eocontrol.EOKeyValueQualifier";
                "value" = "findHouseByAskingPrice";
                "selectorName" = "isEqualTo";
                "key" = "operationName";
            };
        },
```

3. In Rule Editor, click New.
4. Enter `"serviceLocationURL"` in the Key text input field.
5. Enter `(serviceName = 'HouseSearch')` in the Left-Hand Side pane.
6. Enter `http://17.203.33.19:5299/cgi-bin/WebObjects/HousesForSale.woa` in the Value text field.
7. Enter `50` in the Priority text field.
8. Save the `user.d2wmodel` file.
9. Close the Web Services Assistant window.



user.d2wmodel

Lhs	Rhs Key	Rhs Value	Priority
(serviceName = 'HouseSearch')	"serviceLocationURL"	http://17.203.33.19:5299/cgi-bi	50
(operationName = 'findHouseByAskingPrice')	argumentKeys	(low, high)	100
(operationName = 'findHouseByCity')	argumentKeys	(city, state)	100
((operationName = 'findHouseByAskingPrice') and (	comparisonKey	<	100
((operationName = 'findHouseByAskingPrice') and (	comparisonKey	>	100
(operationName = 'findHouseByAskingPrice')	entityName	Listing	100
(operationName = 'findHouseByCity')	entityName	ListingAddress	100
(operationName = 'findHouseByAskingPrice')	isOperationFrozen	false	100
(operationName = 'findHouseByCity')	isOperationFrozen	false	100

31 rules

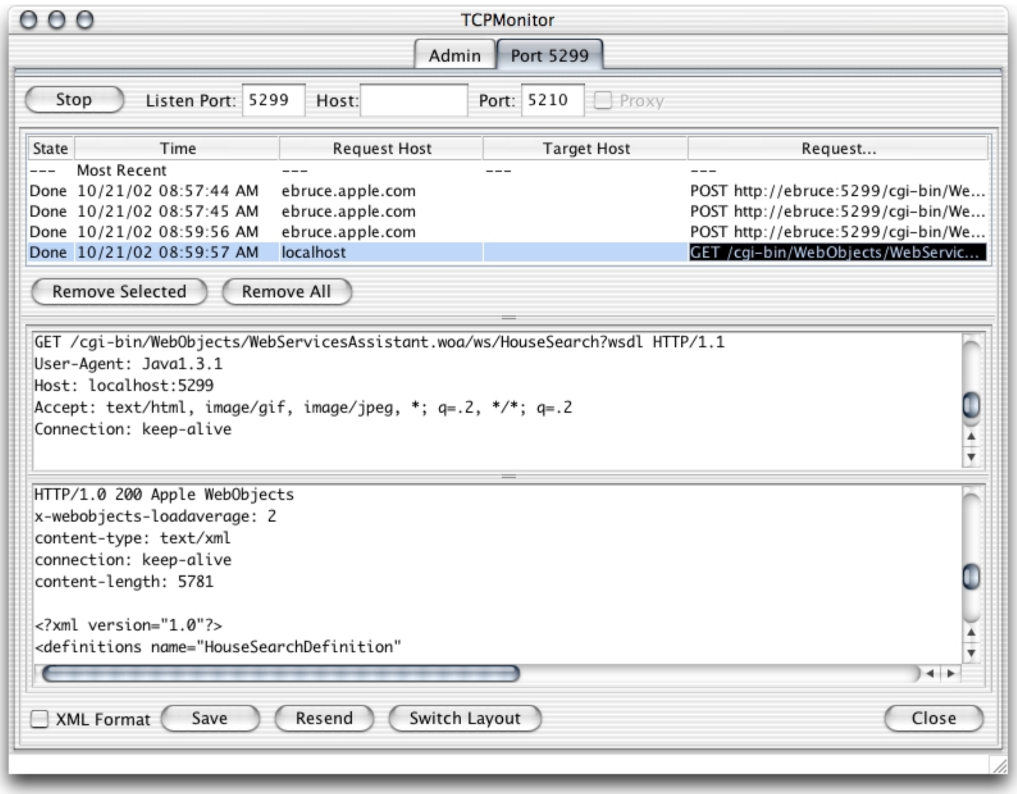
Left-Hand Side: (serviceName = 'HouseSearch')

Right-Hand Side
Class: Assignment
Custom:
Key: "serviceLocationURL" Priority: 50
Value: "http://17.203.33.19:5299/cgi-bin/WebObjects/HousesForSale.woa"

10. Launch TCPMonitor by double-clicking TCPMonitor in `/Developer/Examples/JavaWebObjects`.
11. Enter `5299` in the Listen Port text input field and `5210` in the Target Port text field and click Add.
12. Display the Port `5299` pane of TCPMonitor.
13. In Web Services Assistant, enter `http://localhost:5299` in the text input field of the Connect dialog and click Connect.

    Notice that TCPMonitor shows you the request and response documents as Web Services Assistant communicates with the HousesForSale application.
14. If you test the findByCity or findByAskingPrice operations, the WSDL (Web Services Description Language) document for the HouseSearch Web service is displayed in the response pane of TCPMonitor, as shown in Figure 6-8.

Figure 6-8 TCPMonitor window



State	Time	Request Host	Target Host	Request...
---	Most Recent	---	---	---
Done	10/21/02 08:57:44 AM	ebruce.apple.com		POST http://ebruce:5299/cgi-bin/We...
Done	10/21/02 08:57:45 AM	ebruce.apple.com		POST http://ebruce:5299/cgi-bin/We...
Done	10/21/02 08:59:56 AM	ebruce.apple.com		POST http://ebruce:5299/cgi-bin/We...
Done	10/21/02 08:59:57 AM	localhost		GET /cgi-bin/WebObjects/WebServic...

```
GET /cgi-bin/WebObjects/WebServicesAssistant.woa/ws/HouseSearch?wsdl HTTP/1.1
User-Agent: Java1.3.1
Host: localhost:5299
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
```

```
HTTP/1.0 200 Apple WebObjects
x-webobjects-loadaverage: 2
content-type: text/xml
connection: keep-alive
content-length: 5781

<?xml version="1.0"?>
<definitions name="HouseSearchDefinition"
```

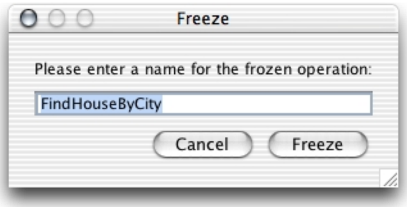
Freezing Operations

You can freeze operations when you need to customize their workings. Frozen operations take the form of Web components in your application project. When you freeze an operation, the parts of the Web service's WSDL document that correspond to the operation are frozen as well. In addition, you cannot use Web Services

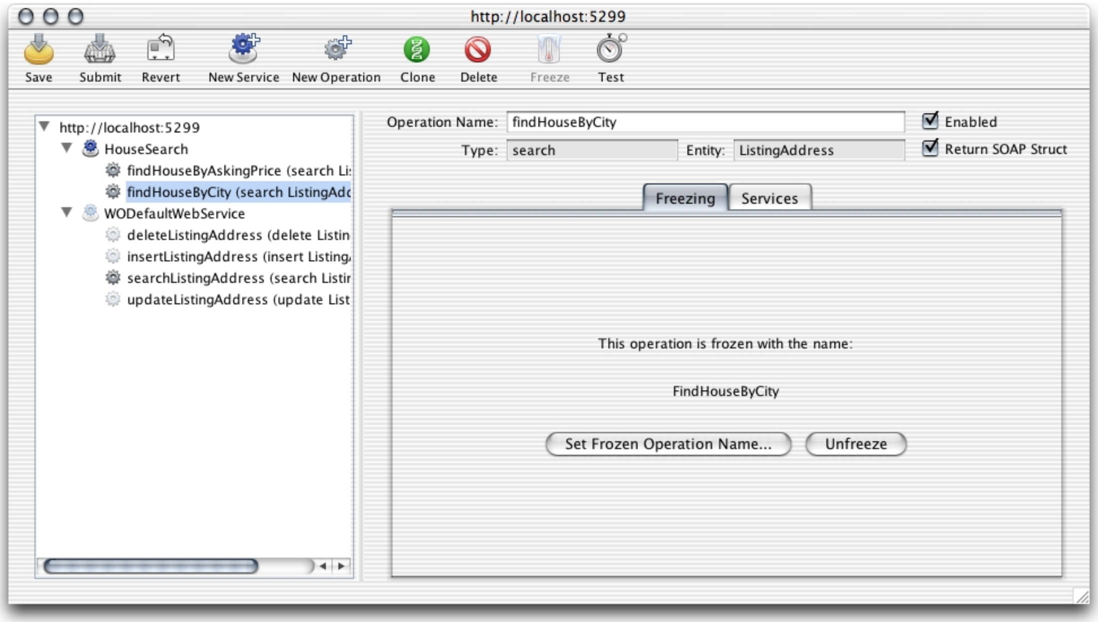
Assistant to customize further a frozen operation; for example, you cannot add or remove arguments or return values with Web Services Assistant. If you need to do so, you have to edit the Java file and WSDL document manually.

The following list itemizes the steps needed to freeze an operation.

1. In Web Services Assistant, select the operation you want to freeze and click the Freeze toolbar button. In the Freeze dialog, enter the name of the frozen-operation component and click Freeze.



Please enter a name for the frozen operation: FindHouseByCity



http://localhost:5299
- HouseSearch
  - findHouseByAskingPrice (search Li
  - findHouseByCity (search ListingAdc
- WODefaultWebService
  - deleteListingAddress (delete Listin
  - insertListingAddress (insert Listing
  - searchListingAddress (search Listir
  - updateListingAddress (update List

Operation Name: findHouseByCity ☑ Enabled

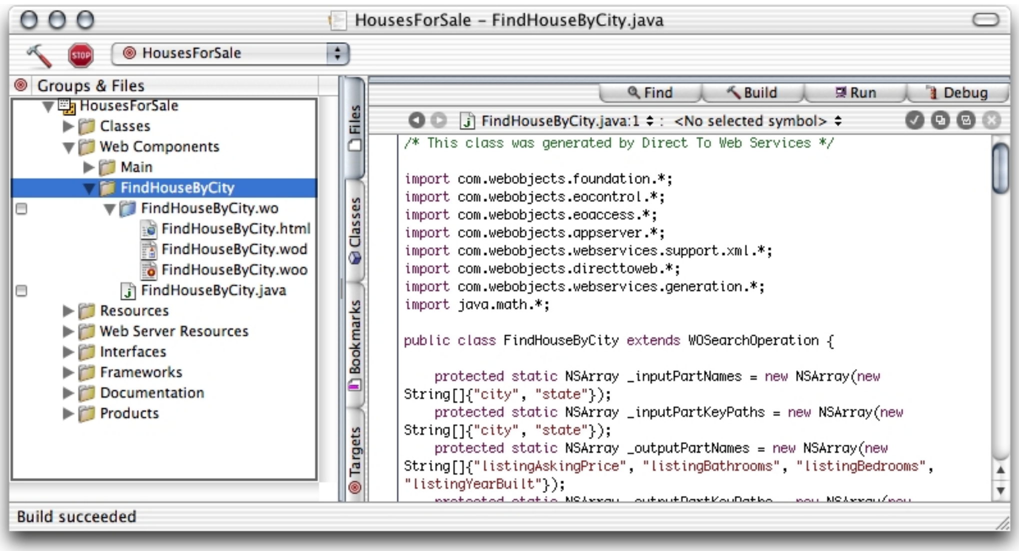
Type: search Entity: ListingAddress ☑ Return SOAP Struct

This operation is frozen with the name:

FindHouseByCity

Web Services Assistant adds the FindHouseByCity component to the HouseForSale project, as shown in [figure].

Figure 6-9 The FindHouseByCity component—the frozen version of the `findHouseByCity` operation



```
/* This class was generated by Direct To Web Services */

import com.webobjects.foundation.*;
import com.webobjects.eocontrol.*;
import com.webobjects.eoaccess.*;
import com.webobjects.appserver.*;
import com.webobjects.webservices.support.xml.*;
import com.webobjects.directtoweb.*;
import com.webobjects.webservices.generation.*;
import java.math.*;

public class FindHouseByCity extends WOSearchOperation {

    protected static NSArray _inputPartNames = new NSArray(new
String[]{"city", "state"});
    protected static NSArray _inputPartKeyPaths = new NSArray(new
String[]{"city", "state"});
    protected static NSArray _outputPartNames = new NSArray(new
String[]{"listingAskingPrice", "listingBathrooms", "listingBedrooms",
"listingYearBuilt"});
```

2. Save the Web service configuration and close the Web Services Assistant window.
3. Restart the application.

The WSDL document corresponding to a frozen operation is stored in the HTML file of the corresponding component. Listing 6-2 shows the WSDL document for the frozen `findHouseByCity` operation.

Listing 6-2 The WSDL document of the frozen findHouseByCity operation—the HTML file of the FindHouseByCity component

```
<?xml version="1.0"?>
<definitions name="[AnyService]Definition"
```

```
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:tns="http://17.203.33.19/cgi-bin/WebObjects/HousesForSale.woa/ws/[AnyService]/
wsdl"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:lang="http://lang.java/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:webobjects="http://www.apple.com/webobjects/webservices/soap/"
    targetNamespace="http://17.203.33.19/cgi-bin/WebObjects/HousesForSale.woa/ws/
[AnyService]/wsdl">
    <types>
    </types>
    <message name="findHouseByCityInput">
      <part type="xsd:string" name="city"/>
      <part type="xsd:string" name="state"/>
    </message>
    <message name="findHouseByCityOutput">
      <part type="xsd:anyType" name="return"/>
    </message>
    <message name="WSDLInput">
    </message>
    <message name="WSDLOutput">
      <part type="xsd:anyType" name="return"/>
    </message>
    <message name="beginTransactionInput">
    </message>
    <message name="beginTransactionOutput">
      <part type="xsd:anyType" name="return"/>
    </message>
    <message name="commitTransactionInput">
    </message>
    <message name="commitTransactionOutput">
      <part type="xsd:anyType" name="return"/>
    </message>
    <message name="rollbackTransactionInput">
    </message>
    <message name="rollbackTransactionOutput">
      <part type="xsd:anyType" name="return"/>
    </message>
    <portType name="[AnyService]PortType">
```

```
        <operation name="findHouseByCity" parameterOrder="city state">
            <input message="tns:findHouseByCityInput"/>
            <output message="tns:findHouseByCityOutput"/>
        </operation>
        <operation name="WSDL">
            <input message="tns:WSDLInput"/>
            <output message="tns:WSDLOutput"/>
        </operation>
        <operation name="beginTransaction">
            <input message="tns:beginTransactionInput"/>
            <output message="tns:beginTransactionOutput"/>
        </operation>
        <operation name="commitTransaction">
            <input message="tns:commitTransactionInput"/>
            <output message="tns:commitTransactionOutput"/>
        </operation>
        <operation name="rollbackTransaction">
            <input message="tns:rollbackTransactionInput"/>
            <output message="tns:rollbackTransactionOutput"/>
        </operation>
    </portType>
    <binding type="tns:[AnyService]PortType"
name="[AnyService]SoapBinding"><soap:binding style="rpc" transport="http://
schemas.xmlsoap.org/soap/http"/>
        <operation name="findHouseByCity">
            <soap:operation soapAction="http://17.203.33.19/cgi-bin/WebObjects/
HousesForSale.woa/ws/[AnyService]/wsdl"/>
            <input>
             <soap:body use="encoded" namespace="http://17.203.33.19/cgi-bin/WebObjects/
HousesForSale.woa/ws/[AnyService]/wsdl" encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/"/>
            </input>
            <output>
             <soap:body use="encoded" namespace="http://17.203.33.19/cgi-bin/WebObjects/
HousesForSale.woa/ws/[AnyService]/wsdl" encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/"/>
            </output>
        </operation>
        <operation name="WSDL">
            <soap:operation soapAction="http://17.203.33.19/cgi-bin/WebObjects/
HousesForSale.woa/ws/[AnyService]/wsdl"/>
```

```
            <input>
             <soap:body use="encoded" namespace="http://17.203.33.19/cgi-bin/WebObjects/
HousesForSale.woa/ws/[AnyService]/wsdl" encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/"/>
            </input>
            <output>
             <soap:body use="encoded" namespace="http://17.203.33.19/cgi-bin/WebObjects/
HousesForSale.woa/ws/[AnyService]/wsdl" encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/"/>
            </output>
        </operation>
        <operation name="beginTransaction">
            <soap:operation soapAction="http://17.203.33.19/cgi-bin/WebObjects/
HousesForSale.woa/ws/[AnyService]/wsdl"/>
            <input>
             <soap:body use="encoded" namespace="http://17.203.33.19/cgi-bin/WebObjects/
HousesForSale.woa/ws/[AnyService]/wsdl" encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/"/>
            </input>
            <output>
             <soap:body use="encoded" namespace="http://17.203.33.19/cgi-bin/WebObjects/
HousesForSale.woa/ws/[AnyService]/wsdl" encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/"/>
            </output>
        </operation>
        <operation name="commitTransaction">
            <soap:operation soapAction="http://17.203.33.19/cgi-bin/WebObjects/
HousesForSale.woa/ws/[AnyService]/wsdl"/>
            <input>
             <soap:body use="encoded" namespace="http://17.203.33.19/cgi-bin/WebObjects/
HousesForSale.woa/ws/[AnyService]/wsdl" encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/"/>
            </input>
            <output>
             <soap:body use="encoded" namespace="http://17.203.33.19/cgi-bin/WebObjects/
HousesForSale.woa/ws/[AnyService]/wsdl" encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/"/>
            </output>
        </operation>
        <operation name="rollbackTransaction">
```

```
            <soap:operation soapAction="http://17.203.33.19/cgi-bin/WebObjects/
HousesForSale.woa/ws/[AnyService]/wsdl"/>
            <input>
             <soap:body use="encoded" namespace="http://17.203.33.19/cgi-bin/WebObjects/
HousesForSale.woa/ws/[AnyService]/wsdl" encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/"/>
            </input>
            <output>
             <soap:body use="encoded" namespace="http://17.203.33.19/cgi-bin/WebObjects/
HousesForSale.woa/ws/[AnyService]/wsdl" encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/"/>
            </output>
        </operation>
    </binding>
    <service name="[AnyService]">
        <port name="[AnyService]Port" binding="tns:[AnyService]SoapBinding">
            <soap:address location="http://17.203.33.19/cgi-bin/WebObjects/
HousesForSale.woa/ws/[AnyService]/wsdl"/>
        </port>
    </service>
</definitions>
```

Unfreezing Operations

To unfreeze a frozen operation follow these steps:

1. In Web Services Assistant, select the operation you want to unfreeze.
2. Click Unfreeze in the Freezing pane.
3. Save the Web service configuration.
4. In Project Builder, delete the corresponding component.
    a. Select the component under the Web Components group.
    b. Choose Edit > Delete.
    c. In the Delete References dialog, click Delete References & Files.
    d. In the Finder, delete the corresponding `.wo` file from the project's directory.

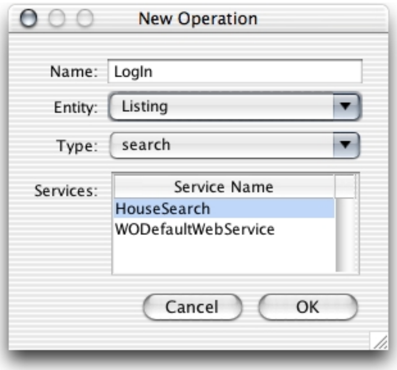
Dynamic Custom Operations

"Freezing Operations" (page 53) indicated that when you freeze an operation, the operation's WSDL document as well as its parameters and return values are cannot be customized using Web Services Assistant.

You can create custom operations whose WSDL document is dynamic. For that you need to copy the `templates/D2WSOperation.pbfiletemplate` to `/Developer/ProjectBuilder Extras/File Templates/WebObjects`.

Follow these steps to create an operation with a dynamic WSDL document:

1. Create an operation in Web Services Assistant.



2. Save the Web service configuration in Web Services Assistant and close the Web Services Assistant window.
3. In Project Builder, select the Web Components group and add a D2WSOperation component with the same name of the operation added in Web Services Assistant.

   To confirm that the component's `invoke` method is invoked, edit the `invoke` method of its Java file so that it like this:

   ```
   public Object invoke() {
       System.out.println("LogIn operation invoked.");
   ```

```
        return super.invoke();
    }
```

4. Add the following rule to the `user.d2wmodel` file:

   ```
   LHS: (operationName = 'LogIn')
   Key: operationClassName
   Value: "LogIn"
   Priority: 100
   ```

5. Save `user.d2wmodel`.
6. Rebuild and run the application.
7. Connect to the application using Web Services Assistant.
8. Test the operation. Make sure that you see the test output in the console (Project Builder's Run pane).