Omnis Web Services

Using the Omnis Web Services Component

TigerLogic Corporation
September 2009
08-092009-02

Table of Contents

About This Manual

This manual describes how you can subscribe to or publish your own Web Services using Omnis Studio, focusing on the creation of client applications that interact with remote Web Services as well as the exposure of functionality within Omnis applications as Web Services.

This manual assumes you know how to create Omnis libraries, add classes and components to your libraries, and write and edit Omnis methods using the method editor. If you are unfamiliar with these procedures you should read the *Introducing Omnis* manual and work through the tutorial.

In addition, you should refer to the *Omnis Programming, Extending Omnis* and *Omnis Reference* manuals, as well as the Help system (using the F1 key), for information about the commands and functions available in Omnis Studio.

Introduction

The concept of a *Service Orientated Architecture* (SOA) represents the latest steps in Business IT to improve the integration of different applications and computing platforms that may exist in an organization or across distributed networks. One of the most successful implementations of a SOA to date is using *Web Services*, and this technology provides many innovative and cost-saving opportunities for medium and larger size enterprises.

Omnis Studio fits very neatly into the SOA concept since it is very adept at integrating different technologies, and allowing developers to provide functionality on many different platforms and environments. With the introduction of the Web Services product, Omnis developers will be able to incorporate tried and tested business functionality, provided as Web Services, into their own Omnis applications. In addition, Omnis developers will be able to exploit their own applications by providing existing functionality to many new markets via Web Services.

About Web Services

What is a Web Service?

The definition of a Web Service is: a standardized way of integrating Web-based applications using the XML, SOAP, and WSDL open standards over an Internet protocol backbone. SOAP is used to transfer the data (via HTTP/S), while WSDL is used for describing the services available. Used primarily as a means for businesses to communicate with each other and with clients, Web Services allow organizations to communicate data without intimate knowledge of each other's IT systems behind the firewall. (Source: www.webopedia.com).

You can find more information about Web Services, and their specification, from:

- www.w3.org
  Includes definitions and all current standards for Web Services
- www.webservices.org
  Information and discussion about Web Services

In the context of Omnis Studio, a *Web Service* can be implemented as one or more Omnis methods within a remote task. Such a web service can be accessed by any other application, including Omnis itself, via direct or remote HTTP access via the Omnis Server and a standard Web Server. In addition, the Omnis Web Services component allows Omnis applications to consume or subscribe to any existing Web Service.

The client and server implementation of the Web Services product uses the Omnis Java Objects interface and fully complies with the SOAP and WSDL standards.

Simple Object Access Protocol (SOAP)

Simple Object Access Protocol (SOAP) is a protocol for exchanging XML-based messages over a computer network, normally using HTTP. These messages take the form of SOAP requests when a client attempts to call a web service and SOAP responses which contain the server‟s return value.

Web Services Description Language (WSDL)

Web Services Description Language (WSDL) is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint.

Service Orientated Architecture

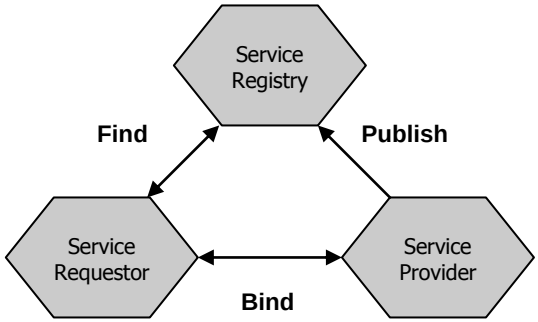


Diagram 1. Diagram of a Service Orientated Architecture

Web Services are an example of a Service Orientated Architecture (SOA) as shown in diagram 1*. The SOA has three main roles and three operations. The three roles are Service Requestor, Service Provider and Service Registry.*

Service Requestor

The Service Requestor is the client side of the Web Service client server relationship. Any consumer of a web service can be considered a Service Requestor. Its main responsibilities are to find services published on Service Registries and to bind to their Service Provider.

Service Provider

The Service Provider is the server side of the Web Service client server relationship. Any provider of a web service can be considered a Service Provider. Its main responsibilities are to create and publish service descriptions on Service Registries and to bind to various Service Requestors.

Service Registry

The Service Registry is responsible for listing the Web Service descriptions. These descriptions are published by Service Providers and can be searched by Service Requestors.

Once the Service Registry has established a match between a Service Requestor and Service Provider it is no longer needed for the interaction between the two entities.

Each of these roles are linked by the three operations of the SOA. These operations are Bind, Find and Publish and Omnis can perform all three operations.

The Find operation

The Find operation is the contract between the Service Requestor and the Service Registry. The Service Requestor states a search criteria, such as a type of service. The Service Registry matches the find criteria against its collection of published Web Services and returns a collection of matched Service descriptions.

The Bind operation

The Bind operation embodies the client–server relationship between the Service Requestor (client) and the Service Provider (server). This takes on the actions of establishing a connection to the Service Provider and sending a SOAP request and then receiving the corresponding SOAP response.

The Publish operation

The Publish operation is a function carried out by the Service Provider. It is a contract with the Service Registry to allow the Provider to publish its Web Service description to the Registry"s collection. This will allow Service Requestors to find the Web Service should they search for the relevant criteria.

About Apache Axis 2

Omnis Studio 5.0 contains a new version of the Web Services component that uses Apache Axis 2. This manual refers to this version of the module not the previous Axis implementation. Note that some editions of Omnis do not allow you to use the Web Services component.

Why upgrade?

Increased Robustness

Firstly, Apache have stopped developing the original Axis component. Any bugs that exist in the old version of Axis will probably remain unfixed. Secondly, the build process has been streamlined significantly; Omnis no longer generates custom code to call the Axis Service Objects. Moving to the Axis 2 based version of the product may address issues encountered with Omnis Web Service Object generation.

New features

The new Omnis Web Services component provides new features, including:

- Support for Complex Type mappings.
- Calls to Web Services persist through a new interface, which take and returns Omnis native types.
- Improved control over Client requests including proxy server and timeout support
- Axis 2 Client Objects contain many new features (http://ws.apache.org/axis2/).

Compatibility with the old Web Services component

Apache Axis 2 is not backward compatible. Web Service Objects created with the original Omnis Web Services component will not work if you upgrade to the new product. In this case, you will need to rebuild your Web Service Objects using the new component. If your Objects contain complex types, you will have written custom Omnis Java Objects code to create and handle these types. This code will have to be rewritten if you choose to migrate to the new version of the component.

Software Requirements

Omnis Studio

To use the latest Omnis Web Services component, you must use the Professional Edition of Omnis Studio 5.0, or higher, and you must purchase a separate serial number to enable the Omnis Web Services plug-in. The new Axis 2 implementation of Web Services provides support for complex types, performance tweaks as well as support for calls via proxy and authenticated proxy servers. However Axis 2 is not backward compatible, therefore if you choose to upgrade to take advantage of the new functionality your clients have to be rebuilt via the Web Service wizard.

Installing the Web Services component

The Omnis Web Services plug-in is available on the Omnis Studio DVD and to download from the Omnis website (www.omnis.net). The OWS installer will install the plug-in and all the other necessary files into the appropriate folders in your Omnis Studio 5.0 product tree. To enable the plug-in, you must purchase an Omnis Web Services plug-in serial number and enter it in Omnis. You should purchase one development serial number (license) per developer, whereas deploying the Web Services component is free of charge.

To serialize the Omnis Web Services component, you must open the Registration Details dialog using the Tools>>Change Serial Number menu option in Omnis, click on the Plug-ins tab and enter the Web Services plug-in serial number. *You must restart Omnis for the plug-in serial number to take effect.*

If the Web Services browser node does not appear in the Studio Browser it could be that the Web Services libraries have not loaded (wsc.lbs and wss.lbs in the Startup folder), or that Omnis has not been serialized correctly. Check the trace log (Tools>>Trace log… option) for possible errors.

Java

The underlying implementation for the Omnis Web Services component requires a Java environment on the client machine. To develop client and server applications that access Web Services in Omnis you need a Java compiler. You will need to install version 1.5.x or higher of the JDK to create Omnis Web Services applications. To test or deploy your application in a JVM environment, you need to install version 1.5.x or higher of the J2RE or J2SE (Java SE) environment on your machine or wherever your Omnis application is deployed.

Environment Variables

To ensure the JVM is setup correctly you have to define the OMNISJVM Environment Variable. This should point to the *jvm.dll* (Windows) or *libjvm.so* (Linux) in your J2RE/J2SE installation directory. On Mac OS X, by default, no additional setup is required for Web Services.

Omnis will place certain Java files/folders into the \java folder within your Omnis folder. If for any reason you move these (not recommended), you will need to set the CLASSPATH or OMNISCLASSPATH environment variable to the location of your Java folder. Moving files located in the \java\lib folder will cause Web Services to stop working.

Under Windows, you can add or edit environment variables in the System Properties dialog in the Control Panel. On the Advanced tab click the Environment Variables button, and then click „New" to define a new variable or select the name of the User or System variable you want to change and click „Edit".

Further details about installing Java and setting environment variables are described in the Java Objects chapter in the *Extending Omnis* manual.

Java cache

When Omnis is started for the first time, it will create a cache file for all Java System classes and any classes in your CLASSPATH automatically. This cache improves performance, as it removes the necessity to interrogate the Java Virtual Machine to find out which classes are available every time Omnis starts up.

Therefore, when you upgrade to a new version of Omnis Studio you may need to edit the location path of the Omnis java folder in your CLASSPATH. In this case, you will need to reset the Java Class Cache in Omnis. If you add the OWS plug-in to an existing Omnis Studio 5.0 product tree, you will need to reset the Java Class Cache.

Resetting the Java cache

To reset the Java class cache, you can either delete the cache file manually or use the *Reset class cache* option in the Studio Browser (click on the Studio option in the tree list of the Studio Browser, then click on the Java option to show the *Reset class cache* option). The cache file is called „jcache1.dat" and can be found in your Omnis Java folder. You need to restart Omnis after deleting the Java cache.

Object Reference variables

You must be familiar with creating and using Object Reference variables, since we strongly recommend you use them rather than standard object variables when accessing Web Services. See the *Omnis Programming* manual about using Object Reference variables and later in this manual about using them in the context of Web Services.

Omnis Server

In addition, if you intend to publish your own web services you must install and deploy an Omnis Server. In this case, you must be familiar with using Omnis Remote tasks, writing Omnis server methods, editing HTML files, and setting up a web server. There is a web server plug-in for each common type of web server, (IIS, Apache, etc.) provided with the Web Services product which you must install as appropriate; see the *Server Setup* section in this manual, and the *Developing Web Applications* chapter in the *Extending Omnis* manual.

Omnis as a Web Service Client

Using the Omnis Studio Web Services product, Omnis can access Web Services, acting as a Client. In this case, Omnis can create the object classes needed in your library to interact with the Web Service (the Bind Operation)

Finding Web Services

To access a Web Service in your Omnis application, you need to know the location of the service and create an interface to it. A web service is specified in a WSDL file which is usually located on a remote server accessible via a URL. This file can be located on your local machine or can be a URL of a WSDL file.

WSDL Files

Files with the WSDL extension contain web service interfaces expressed in the Web Service Description Language (WSDL). WSDL is a standard XML document type specified by the World Wide Web Consortium. The Omnis Web Services product supports the use of version 1.1 of WSDL.

WSDL files are used to communicate interface information between the Service Provider and the Service Requestor. A WSDL description allows a Service Requestor to utilize a web service"s capabilities without knowledge of the implementation details of the web service.

A WSDL file contains all of the information necessary for a client to invoke the methods of a web service:

- The data types used as method parameters or return values
- The individual methods names and signatures (WSDL refers to methods as operations)
- The protocols and message formats allowed for each method
- The URLs used to access the web service

Locating WSDL files

When you want to use an external web service from within your Omnis Studio application, you should first obtain the WSDL file for the service you want to use. For public web services, the WSDL file will typically be available on the web site of the organization that publishes the web service. For private web services, contact the organization that supports the web service to obtain the WSDL file.

There are a number of Web Services available free-of-charge that are useful for testing purposes or for commercial use within your Omnis application. You may be able to find Web Services that are completely and freely available to use, alternatively some may require you to signup to obtain a registration ID, while most web services will be subject to their own terms and conditions of use. The following list[1] contains links to companies or individuals providing general information about Web Services while others provide links to specific web services that you may want to use for testing or for commercial purposes.

- www.xmethods.net
  Useful source of Web Services for testing
- www.webservicex.net
  Source of Web Services
- www.xmlme.com
  Source of Web Services

When you have located the web service (WSDL file) you require, you can copy its location (URL) and create the necessary interface to the service using the Omnis Object class wizard. This is described later in this chapter.

[1] Disclaimer: *TigerLogic Corporation provides this list solely for your information. TigerLogic is not responsible for the content of external internet sites listed here. TigerLogic Corporation does not recommend or endorse any application or web service mentioned herein. TigerLogic Corporation shall not be liable for any direct, indirect, general, incidental, special or consequential damages in connection with the use of this list or any of the applications or web services referenced by the external internet sites listed here.*

Creating Web Services Objects

Overview

In order for an Omnis Studio application to interact with a remote Web Service, you need to create a Web Service Object class in your Omnis library. This object class is a standard Omnis object class that encapsulates a remote Web Service and acts as the interface to the Web Service and its methods. Each operation available in the Web Service is represented as an Omnis public method in the Web Service object class. These methods can be called in the same way as you would call any method in an object instance. You need to create one Omnis Studio Web Service Object for each remote Web Service that you want to use.

To create an Omnis Web Service Object you can use the Object class wizard available in the Studio Browser. The wizard builds the Omnis object class using a WSDL file. You will have to locate this manually pasting the URL of the WSDL file into the wizard or you can browse to a locally stored WSDL file using the wizard.

Using the Web Service Object Wizard

Assuming you know the location URL of the WSDL file you wish to use, you can use the Omnis Object Class Wizard to create your Web Service object, as well as a window to test the object.

To create a Web Service object class

- Click on the library in which you want to create the web service object class
- Click on the „Class Wizard" link in the Studio Browser and then click on „Object…" link
- Click on the „Web Service Object" icon, enter a name for the object class and click the Create button

For example, if you are using the Delayed Stock Quote web service from xmethods.net, name the object class something like „oGetStockQuote" and click on the Create button.

- (Windows & Linux only) On the initial screen you will need to set the path to the installation directory of your JDK (1.5.x or higher) **Note**: If you have installed the Sun SDK you will have to browse to the JDK directory within the SDK installation directory. The JDK home directory should contain a bin subfolder with javac.exe or javac depending on operating system.

You will need to browse to your JDK installation directory before selecting „Next". This is used by Axis 2 to compile the source it generates. Once this is set it is saved from session to session so you should only have to do this once.

- On the next screen, enter the URL or location on disk of the WSDL file; in the case of the Stock Quote web service: http://www.webservicex.net/stockquote.asmx?wsdl (note we cannot guarantee this Web Service will be available or still exists)

(Please note we cannot guarantee the Web Service shown in this screenshot will be available or still exists.)

The WSDL file can exist remotely on a web server or locally on your machine. If the WSDL file exists remotely and is protected using HTTP basic authentication, you will need to specify the username and password in the appropriate fields. In our example, the stock quote web service does not require authentication, so you can leave the username and password fields blank.

There is an option to create a Web Service Form, which is an Omnis window class based on the Web Service specified in the WSDL file. This will allow you to test each of the operations in the Web Service by providing fields allowing you to send the required parameters. This option defaults to true as it is highly recommended that you create this form not only to test a Web Service, but also to help you understand how to use the object class created by the wizard.

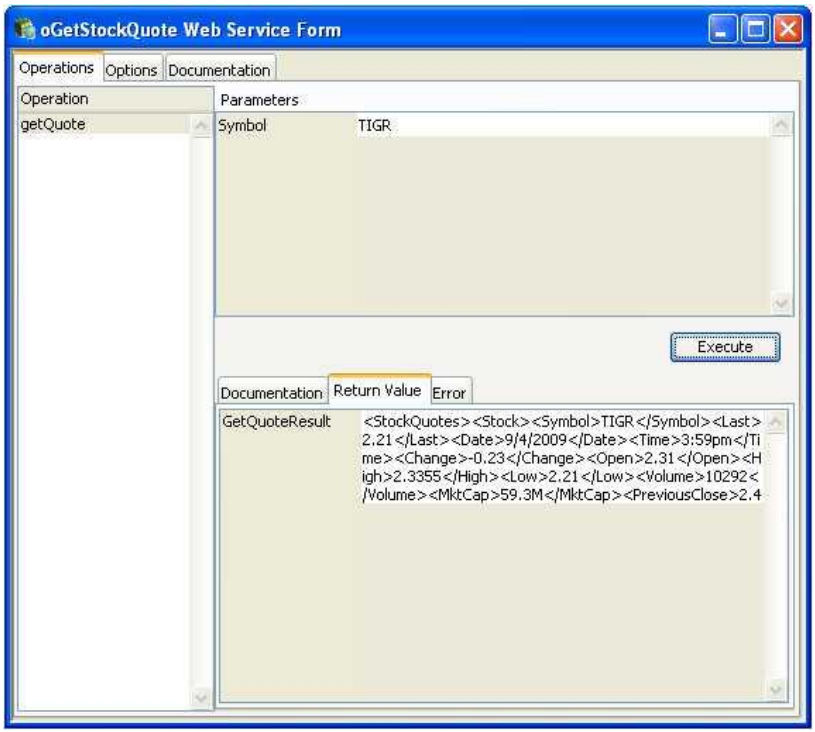
Once you have completed these fields

- Click Next followed by Finish (the Next button may be grayed but click it to activate it)

Note: Some object classes and their underlying Java applications may take several seconds or minutes to compile so do not interrupt the wizard by clicking Close. The time taken to compile the Java application depends on the number and complexity of the operations (methods) in the Web Service.

The Web Service Object wizard builds an Omnis Object class in your library, Omnis Schema classes mapping any complex types within a service, along with the Web Service form (window class). Omnis places the Java application (.jar file) in the \java\webservices folder in your main Omnis Studio folder. When/if you deploy your web service client, you will need to provide the .jar file in the same location on the client machine.

Open the web service window from the Studio Browser and test it using any input parameters as required. In the case of the Stock Quote web service window, you could enter a stock ticker, such as TIGR, and click the Execute button to return the result.



To summarize, the Omnis Web Service object wizard does the following:

- (Windows & Linux) Initializes the path to your JDK for compilation of the Java source code
- Takes the location of the WSDL from the URL you supply (entered manually or browsed to by selecting the „Browse…" button)
- Creates and compiles the Java source code based on the operations in the WSDL file and compiles the code into the .jar application
- Creates an Omnis object class encapsulating the java source file and providing the interface into the remote service; if you checked the create form option, an Omnis window class is created to allow you to test the Web Service object

IMPORTANT: Object reference variables

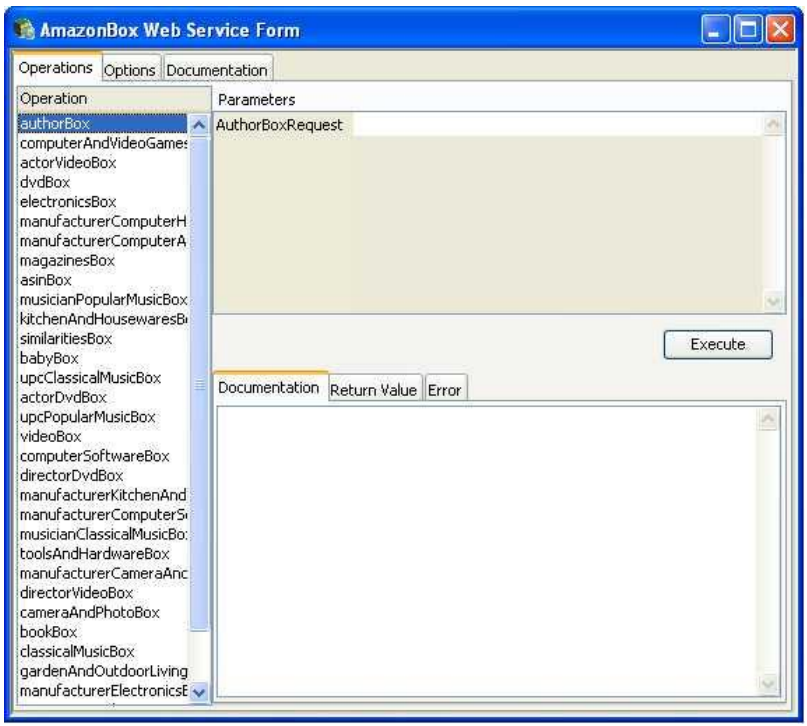
It is highly recommended that the Omnis Web Service Object is used with *Object Reference variables,* since this allows you to use the $deleteref() method of the variable to release the Java Object in the JVM. When the Omnis Object is no longer available it is important that the Java object is de-referenced in the JVM to allow the object to be cleared from memory.

If you instantiate the web service object class as an Object variable, i.e. not a reference variable, a memory leak may occur in the JVM. This is because each Object instance creates a Java counterpart to communicate with the remote Service, and if you use object variables there is no mechanism to clear them out after they have been used.

The Web service form created automatically by the Web Services Object wizard uses Object Reference variables, so you can examine this to find out how you should use object reference variables.

The Web Service Form

During the wizard process of creating a Web Service Object class you have the option of creating a Web Service Form. This is highly recommended as it provides a quick and easy way for you to test the Web Service client that you have just created as well as giving an example of how to interact with the Web Service Object. For example, the following window was created when an Amazon® Web Service object was created using the wizard.



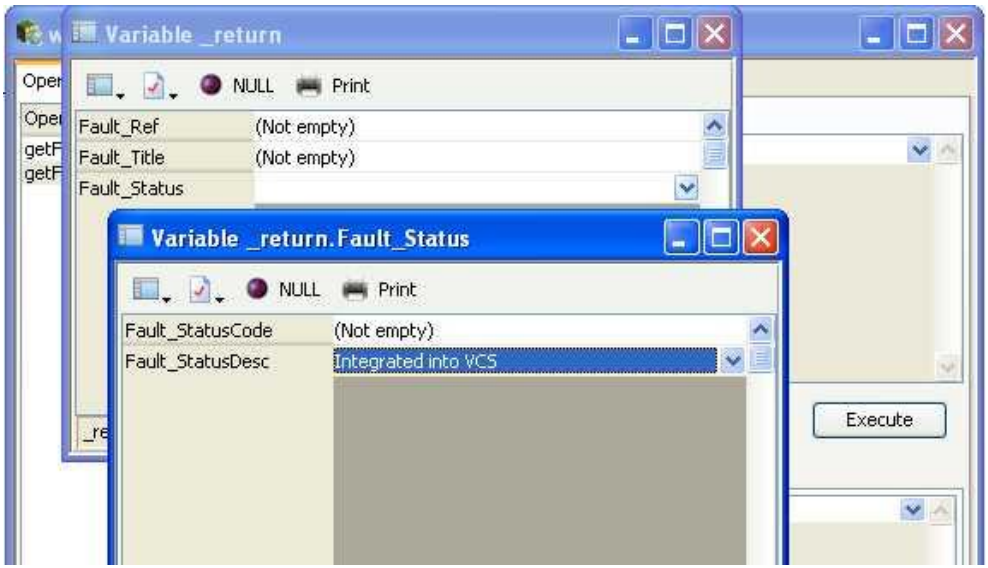
Web Service form tabs

The form has three tabs. The first tab is the main Operations tab that contains the interface required to execute an operation in a web service. The second tab contains settings specific to the Web Service Object as a whole, and the third tab contains any documentation about the service that exists in the WSDL file.

The Operations tab

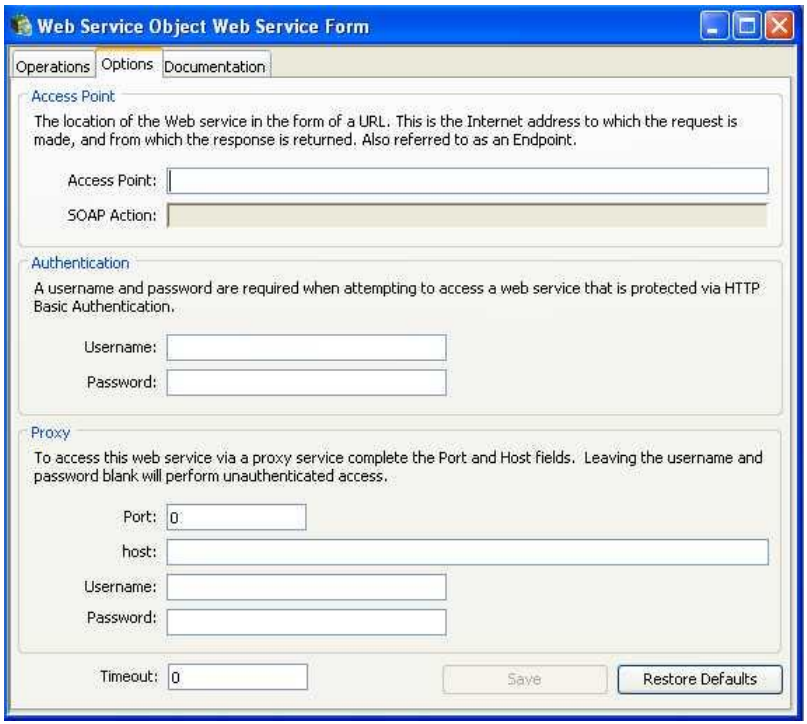
This tab is divided into three main areas. The operations list on the left contains a list of all available operations in the Web Service. Clicking on a line in this list will change the context of the right hand pane. The upper area on the right provides a number of fields, each corresponding to a required parameter of the selected operation. Below these fields is an Execute button. This will call the Web Service passing the values specified in the fields as parameters. The final area, located at the bottom right of the form, contains three further tabs. The Documentation sub-tab displays any information contained in the WSDL for the

selected operation. Upon a successful call to a service invoked by pressing the executed button the tab pane will automatically switch to the Return Value tab. This will display the results of the call. If it has returned a complex type you will be able to expand the rows by clicking on the down arrow on the right of the field. Below shows an example of a multi-layered complex type, the return complex type has three sub types Fault_Ref, Fault_Title and Fault_Status. Expanding Fault_Status presents the Fault_StatusCode and Fault_StatusDesc complex types, containing a single character. FaultStatusDesc is currently highlighted showing the value „Integrated into the VCS".



If the service call is unsuccessful or the call generates a server side error, the tab pane will switch to the Error tab and display details of the error.

The Options tab



The options pane is separated into three groups and an additional timeout field. The first group describes the location of the server and the second group has the authentication details of the service. The third group controls allows the end-user to access the service via a proxy and authenticated proxy servers. The final field allows for adjustment to the default timeout for client requests. Any changes to the fields on this tab should be saved before changing tabs.

Access Point group

The Access Point parameter contains the location of the remote web service. Initially this will be the default value retrieved from the WSDL file. This field should only be modified if the Web Service moves to a new location.

The Soap Action parameter is only available for use with Web Services provided by an Omnis Server. See later in this manual for details.

Authentication group

This group contains the username and password fields that are required by web services that are protected by HTTP Basic Authentication.

Proxy group

This group allows the end-user to access web services via a proxy. Both Port and Host are required to send service requests through a proxy. The username and password field allow authenticated proxy again using HTTP Basic Authentication.

Timeout

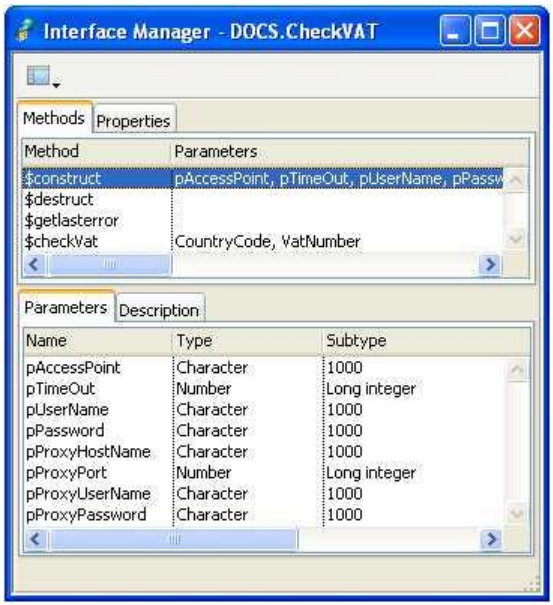
This allows the developer to control the time before a call to a service automatically fails

The Documentation tabs

There are two documentation tabs. The main Documentation tab contains general information found in the WSDL file about the Web Service as a whole. The other one on the Operations tab contains specific information about the selected operation.

Web Service Object Class methods

This section describes the standard methods created in the Web Service object generated by the Object class wizard. When viewing the Web Service Object using the Interface Manager you will see three common methods, $construct(), $destruct() and $getlasterror(),. Also a method for each operation that exists in the Web Service will be displayed. The following object class was created using a check VAT web service; the country code and VAT number is specified in the parameters to the $getRate method. This call will return a Row variable.



$construct

The $construct() method of the Web Service Object is executed on the instantiation of the Object. It has the following eight optional parameters:

- **pAccessPoint** refers to the location of the Web Service. The remote location of a Web Service is usually specified in the WSDL file. If empty the default value stored in the Axis 2 object is used.
- **pTimeOut** sets the time out in seconds for the client request and thus how long Omnis waits for a response from the server. If empty this defaults to sixty seconds.
- **pUserName** and **pPassword** allows the user to send requests via HTTP Basic Authentication to the Web Service. Although optional, both fields are required, else the request will make an unauthenticated call.
- **pProxyHostName** and **pProxyPort** allow the user to make requests via a proxy server, again these are optional but both fields are required .
- **pProxyUserName** and **pProxyPassword** are required if the proxy server is protected by HTTP Basic Authentication.

$destruct

The $destruct() method of the Omnis Web Service Object is called when the Object reference variable goes out of scope. The $destruct() method contains a single call to the $deleteref() method to delete the reference variable. This ensures that the corresponding Java Object is freed in the JVM.

$getlasterror

The $getlasterror() method returns any errors from a call to a Web Service operation. It is recommended that this method is called after each call to an operation.

Operation methods

Each operation that is specified in the WSDL file will have a public method in the Omnis Studio Web Service Object. The name of the method will be the same as the name of the Web Service Operation with „$“ added to the beginning of the name (e.g. $chekVat in the CheckVat service above). The operation methods can be seen in the Omnis Studio Interface Manager along with the required parameters for each method. Where possible the parameters are Omnis variable types, but on some occasions a complex type may be required as a parameter or a return variable.

Complex Data Types

A Web Service Complex type refers to a type that contains one or more sub types. These sub-types in turn can be complex. For instance a WSDL might define an *Address* type that has a number type, *House Number* and a character type, *Road*. *Address* would be an example of a complex type that contains two simple sub-types.

The Axis 2 version of the Web Services component provides support for mapping of complex types into Omnis types. Unfortunately there is a limitation, arrays of complex types as returns and parameters are currently unsupported.

Automatic Handling of Complex Types

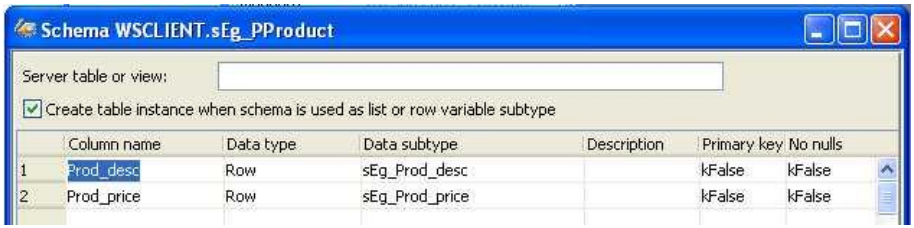
The Axis 2 version of the Web Services Component provides support for automatic mapping of types. The way that complex parameters and complex returns are handled is subtly different.

Complex Types as Parameters

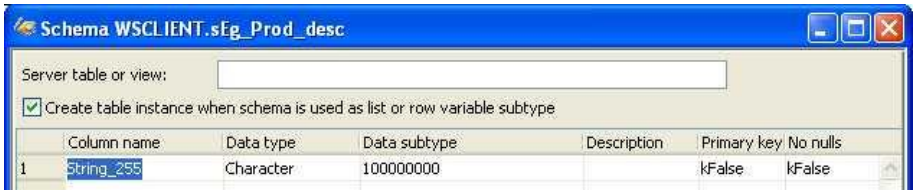
When the Omnis Web Service wizard runs, it will attempt to map complex types to Omnis schema classes containing the sub-types. Consider an Omnis Web Service Method called *addProduct*, which takes a single parameter *Product* that is a complex type containing two sub-types as follows.

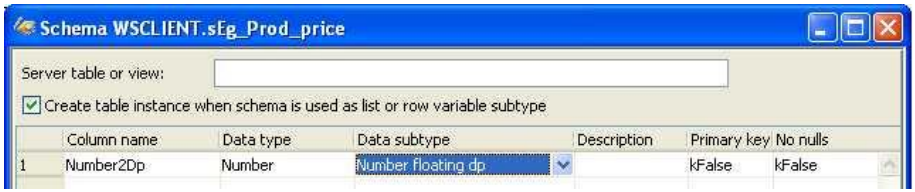
- *prod_desc*, which is a character type with a length of 255 characters
- *prod_price*, which is a number type with a sub type of „Number 2 DP"

If the user saves a WSDL file for this service and builds a new Web Service Object, three new schema classes will be created and placed in a newly created *<object name>_schemas* folder. Firstly, a schema representing the Product parameter (shown below) is created containing the sub-types Prod_desc and Prod_price. These are in turn defined by separate schema classes.



The Prod_desc and Prod_price, although simple, are not standard types thus are also represented by schemas (shown below).





Complex Types as Returns

Consider the opposite of the above example, a *getProduct* method which returns a *Product* type, if we invoke this method the same data structure would be maintained. Invoking *getProduct* service method will return a row variable. The results of such a call are displayed below. As you can see, the return of the method call is returned as a row, called _return, which contains two sub-types *prod_desc* and *prod_price*. These in turn contain a single column row variable that contains our actual data. In the example below the *prod_desc* row variable is displayed with the column name *String_255* containing the character value Camera.

Manual Handling of Complex Types

In some situations Omnis will not be able to fully map complex types; this will result in an empty parameter list on the Web Service Test From. In this situation the developer will have to manually handle complex types. Axis 2 handles this slightly different to the original Axis component, all parameters and returns are declared as inner classes of the stub class. As inner classes are not shown or cached by default due to having a „$" in the class name you will have to move the .jar file associated with the service into the customcalls folder of your webservices folder in the Omnis tree. After this you will need to clear the java class cache and restart studio. See the Deployment section for further details about webservices folder location and resetting java class cache.

After restarting Omnis the object browser should display the inner classes and will allow the developer to make calls using the Omnis Java Objects interface. Please refer to the Java Objects section of the extending Omnis manual for details on making Java calls.

Axis 2 has a standard way of creating its inputs and outputs. It will firstly create a base stub class, this will usually be named <*service name*>Stub and will be the starting point for all service calls. This will contain a number of methods for making service calls, which will match those defined in the Service definition. Each of these methods will usually take zero or one Object parameters. If the method takes zero parameters, the service call takes no parameters. It will usually take a single object parameter in which case you will have to initialize the Object manually. Again Axis 2 has a standard way of doing this and is defined as follows:

- Each parameter set will spawn off of a single java object.
- Each Complex type will have a series of setter and getter methods of the form set<variable name> and matching get<variable name>. These are called to initialize the Object‟s member variables.
- There will be additional getter methods that don‟t have a corresponding getter method and therefore don‟t require calling e.g. $getpullparser, $getomelement, $getobjectvalue
- There can be multiple layers of Objects, i.e. a setter method can actually take a java Object that in-turn has more getter and setter methods.

When you have initialized your parameter data you can call the method. This will return an Object reference containing the data from the call. To get at this data you will need to do the reverse of above by calling the getter methods of the returned object.

Data Type Mapping

The following data type mapping tables show the relationship between the Omnis Studio variables and the Java data types used by the Java client implementation in the Omnis Web Services plug-in.

Java to Omnis Studio data types

Java Type	Omnis Studio Type
char or char[]	Character
byte or byte[]	Binary
int or long	Number Long Integer
float or double	Number Floating dp
short	Number Short Integer
boolean	Boolean
All arrays (except char & byte)	List
java.lang.String	Character
java.util.Date	Datetime Short date
org.apache.axis.types.Time	Datetime Short time
java.util.GregorianCalendar	Datetime (#FDT)
java.lang.Boolean	Boolean
java.lang.Byte	Binary
java.lang.Character	Character
java.lang.Double	Number Floating dp
java.lang.Float	Number Floating dp
java.lang.Integer	Number Long Integer
java.lang.Long	Number Long Integer
java.lang.Short	Number Short Integer
Java Objects	Row
Java Object[]	Not Currently Supported

Omnis Studio to Java data types

Omnis Studio Type	Java Type
Character	char/char[]
Binary	byte/byte[]
Number Short integer	short
Number Long integer	long
Number floating dp	float
Boolean	boolean
Datetime	java.util.Date
List	Can be converted to an array of any Java type depending on list content
Object	Can be converted to an Object of any Java type depending on content
Object Reference	Can be converted to an Object of any Java type depending on the reference

Mapping date and time types

You should try to map date and time data types exactly to avoid misinterpretation of your data. Therefore, Datetime types should be mapped to a variable containing a date and a time. If you wish to return a time, use the Omnis Short time, and similarly, if you wish to return just a Date use the Short date type.

Security

In the Web Services context security means that a message recipient will be able to do some or all of the following:

- Verify the integrity of a message, i.e., that it is unmodified.
- Receive a message confidentially, so that unauthorized parties cannot see it.
- Determine the identity of the sender that is authenticating them.
- Determine if the sender is authorized to perform the operation requested by the message.

Authentication

Determining the identity of the sender and whether they are authorized to carry out the requested operation can be implemented in various ways.

Some Web Services require an identity key or username and password as parameters in the specified operation. The Omnis Studio Web Service Object supports this, as all required parameters by the remote operation would also be required by the corresponding public method in the Web Service Object.

Another form of authentication is protection on the Web Server using HTTP Basic-Authentication. The Omnis Studio Web Service Object supports this type of authentication as the $construct() of the Web Service Object allows users to specify a username and password for use with HTTP Basic Authentication.

Deployment

The Omnis Studio Web Service Object has an underlying Java application associated with it that acts as the bridge between the Omnis Studio application and the remote Web Service. Therefore when deploying an Omnis application that uses a Web Service Object it is important to ensure that the Java application is also deployed.

In the development tree, the Java application is located in the following directory:

`<Studio Root>/java/webservices/`

There will be a single file that contains the Java application located in this folder. The file name will be the same as the name of the Web Service Object, but with the .jar extension. For example, if you have created a Web Service Object called oCurrency, there will be a file called oCurrency.jar in the `<Studio Root>/java/webservices/` folder.

In order for the oCurrency Web Service Object to work in an Omnis Runtime tree, it is important that the oCurrency.jar file is copied to the same folder in the runtime tree. By default the runtime tree does not have a webservices folder, so you will need to create one.

If you have created any custom java code you will also need to mirror your Development tree installation and deploy to `<Studio Root>/java/webservices/customcalls`.

Java cache

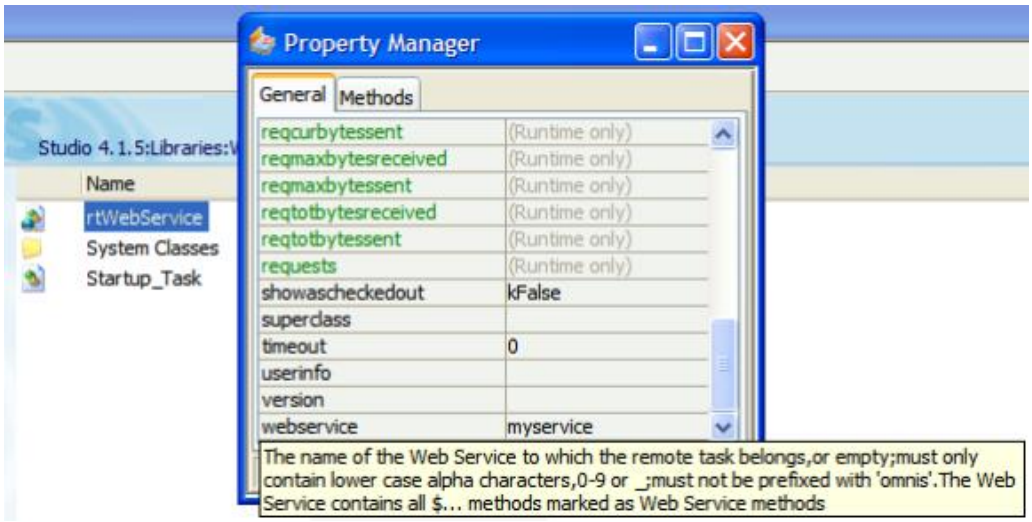
If the runtime tree has been started, a jcache.dat file will have been created in the `<Studio Root>/java/` folder. This file caches paths to the Java classes required by Omnis Studio. If you manually add a Java application to the `<Studio Root>/java/webservices/` folder after this file has been created, you will need to delete the file and restart Omnis so that the application can be located. In addition, if you change the location of the Omnis java folder (not recommended) and/or edit its location in your CLASSPATH, you must reset the Java cache file.

Omnis as a Web Service Provider

Using the Omnis Web Services product, Omnis can implement and publish Web Services by using your code in your Omnis applications. Specifically, a Web Service implemented using the Omnis Server and the Omnis Web Services plug-in allows Web Service clients to call methods in Omnis remote tasks.

Remote Tasks

An Omnis library can contain one or more Web Services. Each of these Web Services contains methods implemented in one or more Omnis *remote task* classes. To identify remote tasks that contain Web Service methods, the remote task class has a property called $webservice.

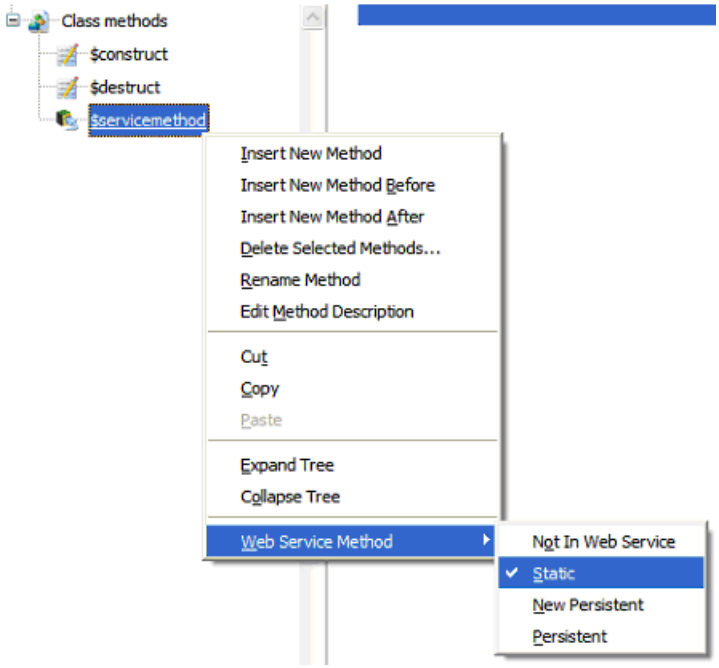


- **$webservice**
  is the name of the Web Service to which the remote task belongs, or empty if the remote task does not belong to a Web Service; assigning a new name to $webservice "creates" the Web Service with that name. The name must only contain lower case alpha characters, 0-9 or „_"; the name must not be prefixed with 'omnis'.

Web Service Methods

All the public methods (any starting with „$“) in the remote task that have been marked as Web Service methods will be available within the web service, with the exception of any methods that have a reserved name.

You can Right-click on a method in the method editor for a remote task and change its status using the „Web Service Method“ context menu.



The Web Service Method menu option is only available if

- You have installed the Omnis Web Service plug-in.
- The method name is not a reserved name; if it is, the menu is replaced by the text “Invalid Web Service Method Name”; reserved names include $construct, $event, and $<java reserved word>.
- The name of the method begins with the $ character.

The Web Service Method menu provides four options:

- **Not In Web Service**
  allows you to exclude a method from the Web Service; for all new and existing methods this is enabled by default, so you must deselect this option to enable a method for a web service

- **Static**
  the method is static, meaning that when a client calls the method, Omnis constructs a new instance of the remote task, Omnis calls the method and then destructs the remote task instance.
- **New Persistent**
  when a client calls the method, Omnis constructs a new instance of the remote task, and then calls the method. The return value from the method includes the connection identifier ($connectionid) of the remote task instance.
- **Persistent**
  the first parameter of the method (as viewed by the client) is the connection identifier of a remote task instance constructed as a result of a call to a method marked as New Persistent. Eventually, a call to one of the Persistent methods in the Web Service must call $cinst.$close() to close the remote task instance, otherwise the remote task instance will persist until it times out, using the normal remote task timeout mechanism.

Methods have a property called $webservicemethod, which can be set using one of the following constants: KWSMethodNone, kWSMethodStatic, kWSMethodNewPersistent, kWSMethodPersistent. These have the same function as the options described above; kWSMethodNone will exclude a method from the Web Service.

Web Service Parameters

In order for clients to use a Web Service developed using Omnis and the Web Services plug-in, they must have a complete definition of all method parameters, and return values, that is, the data types of the parameters and the return values must be fully defined. This enables the Web Services plug-in to properly generate a WSDL file for the Web Service, which is required to build the client. When writing a Web Service method, the following rules apply:

- A Web Service method can have from zero to 32 parameters.
- A Web Service method must return a single data type via one or more calls to the *Quit method* command, that is, the method cannot return a character variable in one case and an integer in another. Each *Quit method* call must use a variable to specify the return value.
- An Omnis parameter cannot have an initial value.

The variable passed to *Quit method*, and the method parameters, can be of any Omnis data type, although the types Picture, Object, Item Reference, Object Reference, and Field Reference are not generally useful, as they do not sensibly map to a standard XML schema data type. Variables of type List and Row *must* have a schema class as their subtype, as described below.

List/Row Subtype

A schema, query, or table class name can be used as the subtype of a list or row variable, that is, a class, instance, local, task or parameter variable, or a column in a list or row defined from a SQL class.

In addition, a schema class has a property $createinstancewhensubtype that controls whether or not there is a table instance associated with a List or Row variable with a schema class as its subtype; you can set this property in the schema editor – it defaults to true for existing and newly created schema classes. When using the schema class exclusively with Web Services, it is likely that the table instance will not be required; turning off $createinstancewhensubtype will therefore improve performance.

Omnis uses the subtype class to define the list or row, or in the case of parameters, to identify the expected definition of the list or row, although Omnis does not do anything if the definition does not match.

WSDL Documentation

Any descriptions you add to your Omnis methods, parameters, schema columns, and so on, are added automatically to the WSDL. This information is included when the client is built using the WSDL file.

Running methods in the background

You may want to be able to run a remote task method in the background to allow the execution of other server methods to continue to run. For example, you could execute a method to print a report in the background while the Omnis Server is continuing to process other data. You can run a method in the background, or asynchronously, using the *Do async method* command.

- **Do async method** *remote-task-class*/*method-name* (*parameters*) Returns *return-value*
  This command uses the Web Services server to execute a method asynchronously in the background, while the user continues to work with the application.

The specified method must have a name allowed for a Web Service method, and it must be marked as a static Web Service method. The method will only execute in the background if you have executed the *Start server* command to start the multi-threaded Web Client server. In a runtime, *Do async method* generates a runtime error if you use it before you have called *Start server*. In the development version, you can omit the call to *Start server* if you wish to debug the method; in this case, the method executes in the foreground, as if it were a normal method call.

The return-value is a long integer that uniquely identifies the call to the method. This is referred to as the „asynchronous call id". You use the asynchronous call id to cancel the asynchronous method with the *Cancel async method* command, and to associate the completion message (see below) with the method call.

Note In line with the requirements for the Web Services plug-in product, the *Do async method* command will run only if you meet some serial number requirements: you need a Web edition serial number, and for the development version, a Web Services serial number.

Passing Parameters

You can include a list of parameters with the *Do async method* command which are passed to the called method. If the called method has fewer parameters than values passed to it, the extra values are ignored.

Completion Message

When the method executing in the background finishes, Omnis sends a message to the task instance that was current when Do async method was called. The message is

```
$asynccomplete(iCallId,cErrText,vRetVal)
```

where iCallId is the asynchronous call id returned by *Do async method*, and vRetVal is the return value of the method executed in the background, unless an error occurred, in which case cErrText is not empty, and contains information about the error.

Example: Background Printing

The following remote task method, $backgroundmethod, runs asynchronously in the background and prints a report, while the completion message sends to the screen. The method is implemented in a remote task, and marked as a Web Service static method.

```
Do async method REMOTETASK/$backgroundmethod ('rReport') Returns
   iCallId
;  Returned long integer iCallId uniquely identifies the method call
```

The $backgroundmethod contains the following code.

```
;  Print the report identified by the parameter to memory, and
   return the resulting report
Calculate $devices.Memory.$visible as kTrue
Do $cdevice.$assign(kDevMemory)
Do $prefs.$reportdataname.$assign(iReport)
Set report name [pReportName]
;  Note that Print report can be used in the multi-threaded Web
   Client server from Studio 4.2 onwards
Print report
Quit method iReport
```

The completion message can be handled in the remote task instance using a custom $asynccomplete() method.

```
;  $asynccomplete(pCallId,pErrorText,pReport) in the task instance
   that was current when Do async method was called
If len(pErrorText)=0
   Send to screen
   Print report from memory pReport
End If
```

Multi-threading

You can only call *Do async method* when running in the normal foreground thread. Background threads will be suspended while a message box is displayed. The background threads only execute when the normal foreground thread is not executing.

The usual restrictions about remote task threads apply, for example you cannot debug a background thread, and you cannot use certain commands when running code in a background thread.

Execution of the remote task method occurs in the context of a remote task instance as usual. This means that the remote task $construct() and $destruct() methods are called before and after calling the specified method, and that the user count for the Web Client server must have an available connection.

Critical blocks

If the library containing the remote task closes before the method finishes, Omnis stops its execution, and does not send the completion message. Note that if the method is in a critical block, Omnis will not stop its execution until it leaves the critical block. Also, execution will only stop after the current command being executed by the method completes.

Only use critical blocks for very short time periods in asynchronous methods, as the user interface will be unresponsive while code is running in a critical block.

Canceling background methods

The *Cancel async method* command allows you to cancel the execution of a method that is executing as a result of a call to the *Do async method* command.

- **Cancel async method** {*id-to-cancel* (*return-value-from-do-async-method*)}
  takes a single parameter id-to-cancel, which is the asynchronous call id returned by Do async method

The *Cancel async method* command sets the flag if it has marked the asynchronous method for cancellation. Omnis only checks to see if the method is marked for cancellation after the completion of each method command, so cancellation may not occur immediately.

If you want to execute a sensitive block of code, which should not be cancelled in this way, you can use the *Begin critical block* and *End critical block* commands around the sensitive code. Omnis will only cancel the method execution when the thread ends the critical block.

If the flag is cleared, either the asynchronous call id is invalid, or the method has finished. After successfully canceling a method call, Omnis still sends the $asynccomplete message, but with an error text parameter that indicates that the call was cancelled.

You can only call *Cancel async method* when running in the normal foreground thread.

XML schema data type mapping

The mapping between Omnis data types, and XML schema data types, is as follows:

Omnis Data Type	XML Schema Data Type	Notes
Character, National	xsd:string	Type restriction for maximum length.
Number (Short Integer)	xsd:int	Type restriction 0..255
Number (Short 0dp)	xsd:int	Type restriction –999999999.. +999999999
Number (Short 2dp)	xsd:int	Type restriction –9999999.99.. +9999999.99
Number (Float dp)	xsd:double	
Number (N dp)	xsd:double	Type restriction implemented via a pattern
Boolean	xsd:boolean	
Date time (kDatetime)	xsd:dateTime	Values passed to the Omnis method are in UTC. A return value of this type must be in UTC*
Date time (kDate)	xsd:date	
Date time (kTime)	xsd:time	Values passed to the Omnis method are in UTC. A return value of this type must be in UTC*
Sequence	xsd:int	
Binary	xsd:base64Binary	
List, Row	Aggregate XML type defined in the <types> section of the WSDL file.	The mapping rules in this table also apply to the members of the aggregate type.
All other types	xsd:base64Binary	

Inheritance

The Omnis Web Services product imposes one restriction on inheritance: remote tasks that belong to a Web Service can have a super-class. However, the remote task and all of its super-classes must belong to the same library.

* The Omnis Web Services plug-in provides functions that enable you to convert between UTC (Coordinated Universal Time) and the local time zone. These are described in the *Time Conversion* section later in this manual

Creating Omnis Web Services

When the Omnis Web Services product is installed, the Studio Browser contains a Web Service Server browser which enables you to manage your own Web Services.

When you select the Web Service Server browser, it displays the list of open Omnis libraries that contain a Web Service. Note that this may be different to the list of libraries displayed in the standard Libraries because, since:

1. Not all open libraries may contain a Web Service.
2. Any open but locked libraries that contain a Web Service will also be displayed.

You will also see two hyperlink options displayed, one for Server Configuration, and the other to open the request log. These links are discussed below.

Expanding the Web Service Server node displays all open libraries that contain a Web Service. When you click on one of these library nodes, the browser displays a list of Web Services in the library. When a library is selected, there is a single hyperlink option that allows you to close the library, unless the library is locked, in which case the option is not available.

If your library or web service name does not appear in the Web Service Server browser, check the following:

- ❑ You have specified the service name in the $webservice property of your remote task; this should comply with the naming conventions as described above.
- ❑ You have enabled your method in the remote task using the Web Service context menu; see the section *Web Service Methods* above for details about enabling your web service methods.

- Select the web service in the Browser and click on Show Error Log to check for any errors.

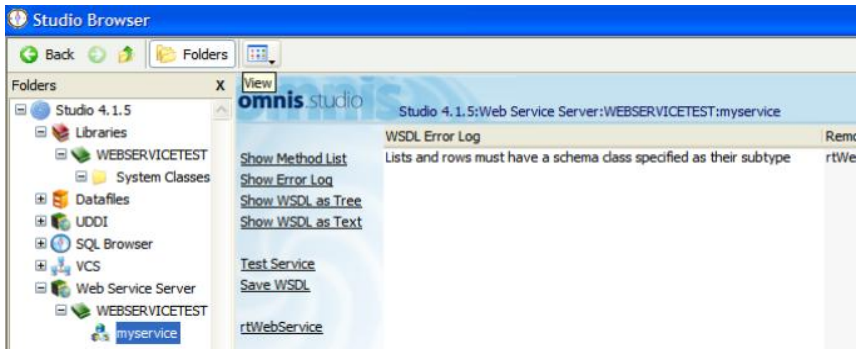
Expanding the library node displays a child node for each Web Service in the library. When you click on one of these child nodes, the browser displays the Web Service and details about its type and respective remote task class.

You can choose one of four views for the Web Service, using the Show… hyperlink options (remember, you can Right-click in the Browser and use Save Window Setup to store the current view).

- **Show Method List**
  The Browser displays the list of methods in the Web Service. If there is a problem with a method marked as belonging to the Web Service, for example a list parameter does not have a schema class as its subtype, the method will not be in this list.
  You can double-click on the method name, to open the method editor for the method, provided that the Web Service does not belong to a locked library.

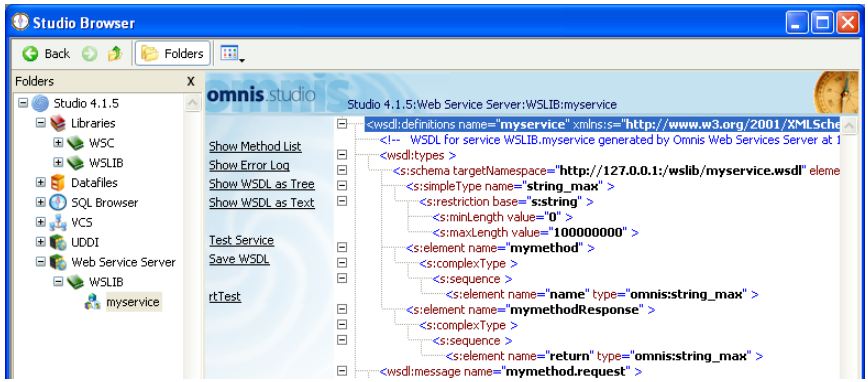
- **Show Error Log**
  The Browser displays the list of errors for the selected Web Service. As you develop your library, you may unintentionally produce method(s) that cannot be included in the Web Service due to an error, for example you might forget to give a list parameter a schema class as its subtype. The Web Services plug-in does not inform you about

errors the moment they occur, rather the error log lists the problems.



If you double-click on the error log line, and the Web Service does not belong to a locked library, the appropriate editor opens at the error. Alternatively, you can use the context menu for the log line, to go to either the error or method.

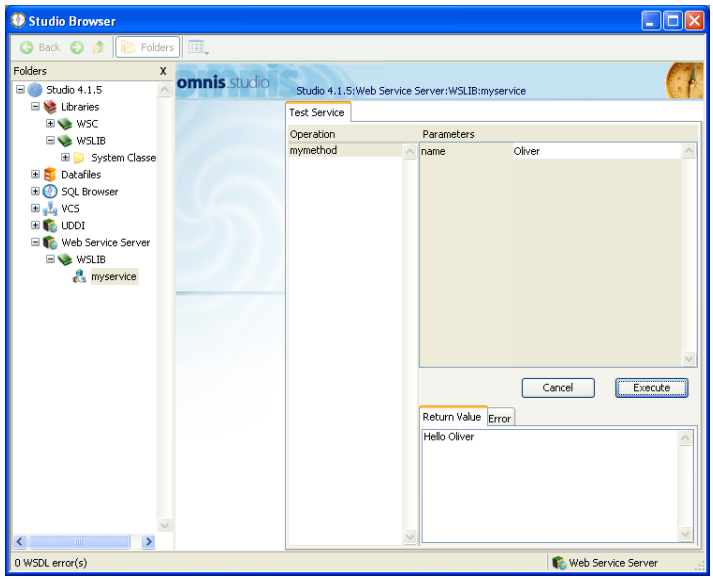
- **Show WSDL as Tree**
  The Browser displays a tree view of the WSDL file for the Web Service. The WSDL will include all methods listed in the method list. There may be a small delay when you select this option, to allow for the WSDL file to be (re)generated.



- **Show WSDL as Text**
  The Browser displays a text view of the WSDL file for the Web Service. The WSDL will include all methods listed in the method list. There may be a small delay when you select this option, to allow for the WSDL file to be (re)generated.

In addition, when a Web Service is displayed, there are additional hyperlink options:

- **Test Service**
  If you select this link, the Browser enters a modal state, and displays a form that you can use to test your Web Service without using a Web Service client:



To test a service/method, select the method in the Operation list, enter the parameter values, and press Execute. The Return Value or error is displayed in the tab pane below. Note that the connection id for persistent methods is managed automatically by

the form, that is, the form remembers the connection id returned by the most recent call to a New Persistent method, and automatically passes it to Persistent methods.
To return the browser to its normal, non-modal state, press the Cancel button.

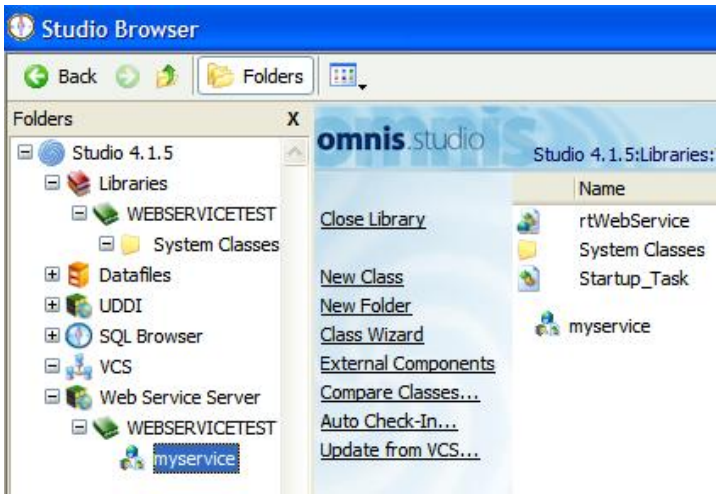
- **Save WSDL**
  Selecting this link prompts you for a name for the WSDL file. You can use this WSDL file for various purposes, e.g. generate a client for the Web Service in another application, or publish the WSDL on a Web Server. However, you are advised to use the Web Service Object wizard to create a client interface for your web service; see the next section about how to do this.
- **<TaskName>**
  A link for each remote task in the Web Service – if the library is not locked, clicking on the link opens the method editor for the remote task.

Creating a Client for your Omnis Web Service

You can generate an Omnis object class for your Web Service (and associated JAR file), as well as a form, to test your web service.

To create a client for your Omnis web service

- Drag your web service from the Web Service Server browser and drop it onto a library (or library folder) in the Studio Browser



A wizard window opens when you drop the service node, which will create the Omnis object class and Java files for the client. You also have the option to create a Form to test your web service.

This is the recommended way of creating the client, because Omnis can generate a client object where the method parameters and return values have exactly the same types as those in the remote task, for example, Omnis list types in the remote task will be matched to list types in the client object. The object generated in this way also includes automatic management of the connection id parameter ($connectionid) for the remote task: it remembers the connection id from the most recent New Persistent method called, and passes it to Persistent methods. If you save a WSDL based on your Omnis web service and then use this to generate the client, types such as lists will be represented as Java objects, which may cause a mismatch.

If necessary, the wizard copies schema classes from the server library to the client library. If there are schema classes needed by the server in other libraries, the client will access them from the same libraries as the server.

Server Setup

You can test your web service using the development version of Omnis but you will need to setup the Omnis Server when you are ready to deploy your Web Service(s). If your clients are going to access your web services via a Web server (remote HTTP access) you will need to install one of the Web server plug-ins designed to work with the Omnis Web Services product.

HTTP Setup

A Web Service built using the Omnis Web Services product uses HTTP as the SOAP transport mechanism. There are two ways that you can use HTTP: Direct HTTP access, or HTTP access via a Web Server.

Direct HTTP access

Omnis Studio includes a basic HTTP server, sufficient to handle Web Service and ultra-thin client requests sent directly to Omnis. The basic server does not offer HTTPS, and you may find that if your server has a high throughput, you still need to use a Web Server instead of the direct server.

To use the ultra-thin client with the direct server, use the following URL:

http://<server>:<serverport>/ultra

where <server> is the domain name or IP address of the machine running Omnis, and <serverport> is the value in the $serverport property of the Omnis server. For example:

```
http://127.0.0.1:5920/ultra
```

HTTP access via a Web Server

This approach supports HTTPS and HTTP authentication (see section below on authentication). HTTP access using an industry standard Web Server may be more suitable for high throughput deployments, as well as those requiring security.

The Web Server approach is very similar to the Omnis Web Client approach. See the *Extending Omnis* manual for more details about setting up the Omnis Server for HTTP web access.

Tomcat web server setup

In order to have the web server function through port 80 (the default port for browsers), the following needs to be added to the server configuration file: \Apache Software Foundation\Tomcat 6.0\conf\server.xml file:

```
<Connector port="80" protocol="HTTP/1.1"
      connectionTimeout="5000"
      redirectPort="8443" />
```

HTTP Authentication

When using a Web Server, you can use any authentication mechanism supported by the client to control access to the Web Server plug-in, provided that the plug-in does not need to participate in the authentication process.

For example, to use basic authentication with IIS 5.1 running on MS Windows XP Professional, you can context click on the scripts directory in the IIS administrative tool, and select Properties. On the Directory Security tab, press the button to edit anonymous access and authentication control, and a window opens. On that window, uncheck the anonymous access checkbox, and check the basic authentication checkbox; enter a default domain and realm, e.g. “omnis” and “webservices”. A client must then use basic HTTP authentication to access the Web Server plug-in in the scripts directory – it will need to supply a user name and password for the XP system. You can try this with the client form created for a Web Service, by entering the user name and password in the Authentication group box on the Options tab.

HTTP GET requests

The Web Services web server plug-in can handle HTTP GET requests, where the URL has the syntax:

http://serveraddress/scripts/wsplugin/library/service.wsdl?OmnisServer=omnisserver

where omnisserver has the same syntax as that used for the web client, so it can include an IP Address, and a load sharing pool if necessary. For example:

```
http://127.0.0.1/scripts/owsisapi.dll
                /library/service?OmnisServer=5920
```

and

```
http://127.0.0.1/cgi-bin/nph-owscgi.exe
                /webservices/service.wsdl?OmnisServer=5920
```

Web Server plug-ins

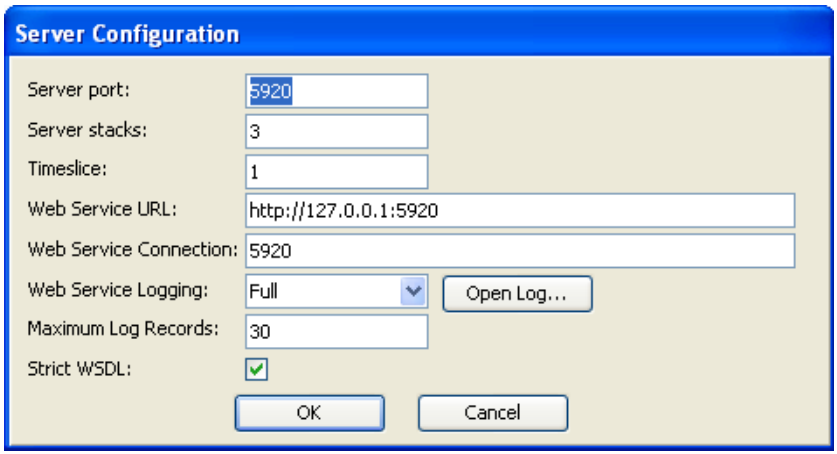
To use Omnis as a Web Service Provider, you must install one of the following plug-ins into your web server, for example, in the /cgi-bin, /scripts, or /webapps folder depending on your platform. Web Server plug-ins are available for ISAPI for Win32, for CGI, and Apache.

Plug-in	Platform(s)	Notes
owsisapi.dll	Win32	This is an ISAPI extension for IIS.
nph-owscgi	Win32, Mac OS X, Linux	This is a CGI executable
mod_ows.so	Win32, Mac OS X	This is an Apache module. Available for Apache 2 on Win32, and Apache 1 and 2 on Mac OS X. Not available for Linux as there is an XML parser incompatibility between the module and Apache. Configuring the module is very similar to mod_omnis.so (web client server plug-in), except that the source file name used for the Apache 1 configuration is owsapach.cpp. You are recommended to use the path /ows_apache to map to the module.

As far as the Web Server is concerned, you can configure and use these in an identical way to the Omnis Web Client Web Server plug-ins. You can also use the load sharing process in conjunction with them.

Omnis Server configuration

If you select the Web Service Server node in the Browser, and click on the Server Configuration hyperlink option, the server configuration dialog is displayed. The options in this dialog correspond to their equivalent notation properties of the Omnis Server stored in $root.$prefs.



Note that the Server Configuration dialog in the development version of Omnis Studio and the Omnis Server (runtime) are identical.

The **Server port** property ($serverport) specifies the port number or service name for the Omnis Server. The port number must be an available TCP/IP port number on the Omnis Server. The number can be between 1 and 32767, but you must not use 80 or any other number used for e-mail, FTP, or other services. You should use a high number, preferably a four or more digit number, for example, 5912. Requests to a Web Service (whether direct, or via a Web Server) arrive at the same port.

The **Server stacks** property ($serverstacks) lets you specify the number of method stacks (and therefore threads) on the Omnis Server, which will be created when you execute the *Start server* command. The **Timeslice** property ($timeslice) lets you specify the duration (in 1/60th second units) of the execution time slice for a server thread.

The **Web Service URL** property ($webserviceurl) tells the client how to connect to the server. It has one of two forms:

1. When using direct HTTP, http://address:port, where address is the domain name or IP address of the machine running Omnis, and port is the server port ($serverport) property, e.g. http://127.0.0.1:5920
2. When using a Web Server, http://address/pluginurl, where address is the domain name or IP address of the Web Server, and pluginurl is the path to the Web Server

plug-in on your Web Server. For example:
http://www.mydomain.com/scripts/owsisapi.dll

The Web Service URL can also be accessed via the notation $root.$prefs.$webserviceurl.

The **Web Service Connection** property ($webserviceconnection) is only needed when using a Web Server. Omnis uses it to populate the SOAPAction header via the WSDL. It contains information that tells the Web Server plug-in how to connect to Omnis. It has the format:

```
[POOL,][IPADDRESS:][PORT]
```

POOL (optional) is a load sharing process pool name, which you only supply when using load sharing; IPADDRESS (optional) and PORT identify Omnis or the load sharing process; if you omit the IPADDRESS, the Web Server and Omnis (or load sharing process) must be running on the same machine. The Web Service Connection can also be accessed via the notation $root.$prefs.$webserviceconnection.

The web service logging fields are discussed in the next section.

The **Strict WSDL** property ($webservicestrictwsdl) determines how WSDLs are interpreted. If checked (set to true, the default), Web Service WSDLs are strict and types are qualified with schema restrictions where possible, and annotations can be added to schema types. You can uncheck this option (set it to false) if your web service client objects to this additional information in the WSDL.

Turning the $webservicestrictwsdl preference off means only the basic schema types in the WSDL are used (rather than qualifying the lengths of strings or the valid range for short integers using our own derived schema types), and the documentation via annotations in the types section of the WSDL are not included.

Server Logging

If you select the Web Service Server node in the Studio Browser, and click on the Open Log hyperlink option, the Web Service Server log is displayed. Alternatively, you can use the Open Log button on the Server Configuration dialog; this provides a way to open the log in the Omnis Server runtime.

The log records requests to any Web Service(s) accessed via the Omnis Server. The log file resides in the Studio folder of the Omnis Server tree, and is called wsslog.df1. You can use the Server Configuration dialog to set the level of logging. It can be one of the following:

1. Nothing (Off)
2. Requests that generate an error (SOAP fault), (Faults)
3. All requests (Full)

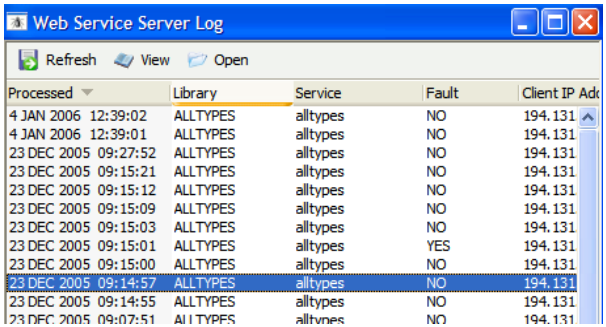
Note: if the Web Server plug-in rejects a request (for example, due to an authentication failure), there will be no record in the log.

You can also use the Server Configuration dialog to set the maximum number of log records.

Alternatively, you can use the Omnis Root ($root.$prefs) properties $webservicelogging and $webservicelogmaxrecords to set the level of logging.

There are two other notation items related to logging:

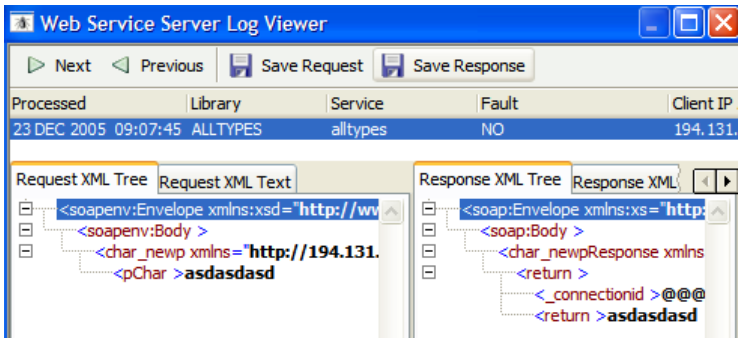
1. $disablewebservicelogging is a library preference ($clib.$prefs). If true, requests to Web Services in the library are never logged. You would typically use this with a locked library, when you do not want the user to use the log to gain access to sensitive data.
2. $openwebservicelog() is a method of $root.$prefs. Executing $root.$prefs.$openwebservicelog() opens the log window.



You can sort the log by clicking on the column headers. The log window has three menus:

- **Refresh**
  Refreshes the log from the data file. For performance reasons, the log window does not dynamically update.
- **View**
  Opens the selected log record in a viewer window (see below). If the record has been deleted because of the maximum of log records property (the oldest record is deleted first), then the viewer window displays a message to that effect.
- **Open**
  Opens an alternative log file.

The Viewer window allows you to look at the request and response in both XML tree and text views. You can also use the Next and Previous buttons to step through the log records in their current sort order (according to the column headers), and to save the request or response to a file.



Handling Times

There are a number of Omnis functions for handling time zones and time conversion. Note these functions are only available in Omnis when the Web Services plug-in is installed and serialized for use. In addition, these functions rely on the Omnis Studio Java Objects implementation, so Java must be available in the Omnis Studio environment.

Time Zones

tzcurrent()

tzcurrent()

Returns the short character identifier for the time zone of the current date and time.

tzdaylight()

tzdaylight()

Returns the short character identifier for the daylight saving time zone.

tzstandard()

tzstandard()

Returns the short character identifier for the standard time zone.

Time Conversion

The Omnis Web Services plug-in provides two functions for converting date-time or time values, between UTC (Coordinated Universal Time) and the local time zone.

loctoutc()

loctoutc(datetime)

Converts the *datetime* (or just a time) from the local time zone to UTC, and returns the result.

utctoloc()

utctoloc(datetime)

Converts the *datetime* (or just a time) from UTC to the local time zone, and returns the result.

The local time zone will be that of the JVM (Java Virtual Machine). This normally corresponds to the time zone of the machine running Omnis Studio (for development) or the Omnis Server. However, you can use the Java options in the Omnis Preferences to set the time zone of the JVM, by setting $usejavaoptions to kTrue, and passing the following to the $javaoptions property.

```
-Duser.timezone=<zone name>
```

For example, set the $javaoptions property to:

```
-Duser.timezone=PST8PDT
```

which sets the time zone of the JVM to the Pacific time zone of the US.

If you pass just a time value to these functions, it will be converted using today"s date. This may result in an anomaly if you use the function for a time around the point where the time changes to or from daylight saving time.

Index