I n s i d e   M a c   O S   X

Web Services

October 2002

Contents

Figures, Listings, and Tables

C H A P T E R 1

Introduction

Welcome to Web services frameworks in Mac OS X version 10.2.

This and subsequent chapters in this book introduce you to Web services available on Mac OS X. Some of the important concepts and terminology behind this newly emerging technology are discussed in this chapter.

The toolkits and frameworks, including Apple's `WebServicesCore.framework`, a client-side framework for accessing Web services from Mac OS X which is new in Mac OS X version 10.2, are discussed in Chapter 2, "Tasks" (page 17). Some of the tools and techniques for writing Web services glue and adding it to Cocoa, Carbon and AppleScript applications are also discussed in Chapter 2.

Finally, Chapter 3, "Web Services Reference" (page 29), provides a reference of APIs, constants, and types that are new in the Web services framework.

Defining Web Services

Web services can be defined broadly as a software service that you can access over the Internet to get a particular result, using standard Web protocols such as XML and HTTP. You use Web services to perform an action on what is defined as an **endpoint**, usually an HTTP server, in order to get the desired results. Web services are the basis of distributed computing on the Internet and offer many opportunities for Web developers.

Web services are *heterogeneous*, in that clients and servers can be running on different architectures and platforms but are still able to speak the same the lingua franca, which is XML. This is one of the major advantages of the Web services architecture.

Some examples of Web services include

- Commercial, fee-based Web services, where in exchange for specific information, the customer pays a prescribed fee. If you have a business that needs some piece of information, for example, a developer could put up a Web service that provides that information. As a customer, you access the information through a Web services toolkit, and are billed for it.
- Corporate intranets, where an Information Services department within the company publishes a Web service and expects their tools to work with it.
- Non-commercial Web services that are publicly accessible. Many of these are public endpoints that companies and Web developers provide for free. For example, Barnes & Noble has a Web service on the Internet, where you give it an ISBN number and it gives you the price of the book. There also are many Web services that provide people with stock quotes, the correct temperature in a zip code, sports scores, or road conditions, as well as other services.

XML-RPC

A remote procedure call (RPC) is a request to a server application at another location to perform operations and return information. XML-RPC is a simple protocol that allows software running in different environments to make remote procedure calls over the Internet. XML-RPC uses two industry standards: XML (extensible markup language) for encoding messages, and HTTP (hypertext transfer protocol) for transporting them. A properly formatted XML-RPC message is an HTTP POST request whose body is in XML. The specified remote server executes the requested call and returns any requested data in XML format.

XML-RPC recognizes procedure parameters by position. Parameters and return values can be simple types such as numbers, strings, and dates, or more complex types such as structures and arrays. To learn more about XML-RPC messages, see the XML-RPC specification at http://www.xmlrpc.com/spec.

The specification defines an XML-RPC message as an HTTP-POST request. The body of the request is in XML. A procedure executes on the server and the value it returns is also formatted in XML. Procedure parameters can be scalars, numbers, strings, dates, etc.; and can also be complex record and list structures.

SOAP

SOAP (Simple Object Access Protocol) is an RPC protocol designed for a distributed environment, where a server may consist of a hierarchy of objects whose methods can be called over the Internet. A goal of SOAP is to establish a standard protocol that will serve both web service providers and service users. As with other remote procedure call protocols, SOAP uses XML to encode messages and HTTP to transport them. A SOAP request contains a header and an envelope; the envelope in turn contains the body of the request.

One key difference between the SOAP and XML-RPC protocols is that with SOAP, parameters are notational (a request must encode the method parameter names within its XML), rather than positional (recognized by position). To learn more about SOAP messages, see the SOAP specification at http://www.w3.org/TR/.

Remote procedure calls provide a powerful tool for accessing services over the Internet. For example, there are already a variety of web-based servers that can check spelling, translate text between languages, provide stock prices, supply weather and traffic information, and more. You can find some available services at sites such as XMethods at http://www.xmethods.net/. There you can also find information you'll need to make remote procedure calls to these services.

Starting with Mac OS X version 10.1, AppleScript and the Apple Event Manager provide XML-RPC and SOAP support such that:

- Scripters can make XML-RPC calls and SOAP requests from scripts.
- Developers can make XML-RPC calls and SOAP requests from applications or other code by sending Apple events.

WSDL

The Web Services Description Language (WSDL) is an XML-based language used to describe the services a business offers and to provide a way for individuals and other businesses to access those services electronically. WSDL is the cornerstone of the Universal Description, Discovery, and Integration (UDDI) initiative spearheaded by Microsoft, IBM, and Ariba. UDDI is an XML-based registry for businesses worldwide, which enables businesses to list themselves and their services on the Internet. WSDL is the language used to do this.

WSDL is derived from Microsoft's Simple Object Access Protocol (SOAP) and IBM's Network Accessible Service Specification Language (NASSL). WSDL replaces both NASSL and SOAP as the means of expressing business services in the UDDI registry.

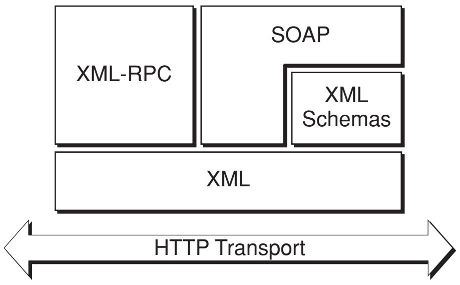
Web Service Invocations

XML-RPC and SOAP are formats for Web service invocations. These are mutually exclusive, just different ways of packaging up the data that you are sending to the Web service. Both are built on top of XML, which is the standard for presenting information in a way that is reliable, guaranteed to understand the message or not, and also very easy for humans to read. These are built on top of HTTP; Web services are typically called or made with HTTP-POST.

In the diagram in Figure 1-1, SOAP sits on top of XML schemas, which gives you a very rich type system.

XML riding on top of HTTP is typically what we think of as a Web service.

Figure 1-1 XML-RPC and SOAP encodings on top of XML on top of HTTP



Section 5 Encoding

In this type of encoding, the XML that you're sending is strongly typed on the XML tag, which specifies, for example, that this element is an integer, or this element is a string. XML schemas let you build up complex types.

The State of XML Web Services

In the Userland XML-RPC specification, XML-RPC is used to implement a simple RPC mechanism on top of XML on top of an HTTP post. This specification is complete; you can implement it today, and is guaranteed to work tomorrow.

SOAP, however, is an evolving standard. The syntax is a little different from XML-RPC. As SOAP evolves, more and more extensions have been added onto it. SOAP 1.1 is a w3c recommendation, which means that it is essentially done. SOAP 1.2. is still a moving target, and its extensions are not yet final. Much of the work on these extensions has been spearheaded by Microsoft and IBM.

In addition, SOAP includes a header, which can contain additional information for SOAP extensions; these extensions can be anything that a service vendor might want to specify.

XML-RPC vs. SOAP

SOAP and XML-RPC are related protocols in that they both attempt to homogenize the passing of parameters to Web services. SOAP is currently a W3C recommendation and is actively being enhanced and developed by Microsoft and other companies. The Microsoft .NET initiative is largely driven by being able to access Web services transparently through SOAP messsages.

XML-RPC is a final specification which is less verbose and easier to implement than SOAP. Both SOAP and XML-RPC work by turning a set of parameters (scalars, strings, dates, arrays, records, and binary data) into XML for transmission. XML-RPC is defined as operating over an HTTP connection, while SOAP describes the envelope format for an RPC request which may be sent over HTTP, SMTP or some other protocol.

SOAP passes parameters by name and XML-RPC passes parameters by position. This is relevant because a routine that depends on the order of parameters in XML-RPC must be called carefully to ensure correct results.

SOAP allows for user record types by extending the XML document using XML Schemas. XML-RPC only allows for the base types defined in the specification. (In reality, any type can be defined using those primitives.)

Both SOAP and XML-RPC support passing binary data in an XML document using Base-64 encoding. XML-RPC has a major flaw, however, in that it defines string parameters as being ASCII text. Some XML-RPC servers will enforce this, forcing the user to pass internationalized text as Base-64 encoded data.

XML-RPC Specification

The complete specification is available at

http://www.xmlrpc.com/spec

SOAP 1.1 W3C Note

SOAP is transport-agnostic, meaning that it is an envelope format but not the transmission. It has recommendations if you send it over HTTP, such as additional headers that you might add, or how it might deal with the server in terms of what error codes it might return. But the SOAP message format itself is the same.

The specification is available at

http://www.w3.org/TR/SOAP/

The schema is available at

http://schemas.xmlsoap.org/soap/envelope

http://schemas.xmlsoap.org/soap/encoding

WSDL Specification

WSDL defines an XML-based grammar for describing network services as a set of endpoints that accept messages containing either document-oriented or procedure-oriented information. The WSDL specification index page links to the WSDL 1.1 specification, related schema, and an overview of the specification.

The specification is available at

http://www.w3.org/TR/wsdl

WSDL Schema

The WSDL Framework is available at

http://schemas.xmlsoap.org/wsdl/

The WSDL SOAP binding is available at

http://schemas.xmlsoap.org/wsdl/soap

The WSDL HTTP GET & POST binding is available at

http://schemas.xmlsoap.org/wsdl/http/

The WSDL MIME binding is available at

http://schemas.xmlsoap.org/wsdl/mime/

Developer Resources

Apple provides a number of resources available to assist developers. These include

- The Apple Developer Networking Documentation at http://developer.apple.com/techpubs/macosx/Networking/
- Apple Sample Code which is available at http://developer.apple.com/samplecode/

C H A P T E R 2

Tasks

This chapter discusses how you can take advantage of Web services that are available in Mac OS X and introduces a new framework that supports WebServicesCore inside of Core Services framework.

Web Services Toolkits

There are currently more than 50 different implementations of XML-RPC as well as more than 80 implementations of SOAP available for developers. Many of these implementations are provided in toolkits for developers. These XML-RPC and SOAP toolkits typically work by binding a language runtime to a serialization format. The required toolkit would match your object model, in most cases.

If, for example, you have an application written in Java, you would likely use one of the established Java SOAP or XML-RPC implementations because those toolkits will match your object model. Thus, the most useful Web services toolkit is one that best matches your runtime environment and your application framework.

Web Services Support in Mac OS X (10.1 or later)

SOAP 1.1 and XML-RPC support are provided in Mac OS X (10.1) via Apple events—the same Apple events that you can use, for example, to script the Finder. Both XML-RPC and SOAP 1.1 are, in effect, "baked into" the Apple Event Manager. Because this support is right there in the Event Manager, you also get AppleScript support "for free."

This is a very popular high-level way for developers on the Macintosh platform to access corporate and public Web services. One distinct advantage is that you don't have to go on the Internet and download a toolkit and then make sure that your customers have it.

An Apple event is an interprocess communication method. The support for Web services works by "hijacking" an addressing mode—`typeApplicationURL`—for an Apple event. In Mac OS X (10.1 or later), using a remote Apple event you can send a binary method to another application on another Mac OS X computer, and for addressing modes which are HTTP, these methods are treated as a Web service.

This is accomplished without any API changes to the Apple Event Manager, and just by enabling some additional data types that are hinted to the Apple Event Manager that the method to be called would use the specified SOAP framework.

Canonical Example: XML-RPC over Apple Event

The canonical XML-RPC example, with both message and response, is quite simple:

1. You call a method, `example.getStateName`, on the server <http://betty.userland.com:80/RPC2>.
2. When you go to send this message, you open an HTTP connection.
3. Then do an HTTP post to that site with an XML document. The XML document takes a single integer with a parameter and returns a string.

Sending a Message

The XML structure shown in Listing 2-1 has both a method name <examples.getStateName> and a parameter.

The element tag is `<i4>`, which indicates that it is typed as a long integer.

The server comes back and specifies that the return value from that method is the string South Dakota. The server is returning an array of 50 states and element 41 in the array in no particular order.

Listing 2-1 An XML structure specifying a method name and parameter

```
<?xml version="1.0" encoding="UTF-8"?>
<methodCall>
    <methodName>examples.getStateName</methodName>
    <params>
        <param>
            <value>
                <i4>41</i4>
            </value>
        </param>
</params>
</methodCall>
```

Getting a Response

Listing 2-2 An XML structure with the return value specified

```
<?xml version="1.0"?>
<methodResponse>
    <params>
        <param>
            <value>South Dakota</value>
        </param>
    </params>
</methodResponse>
```

Using Apple Events to Target a Website

The process of using Apple events to target a website is straightforward, as described by the code example in this section. You would simply follow these steps:

1. Create the parameter list for the method calls. In this example, you create a parameter list with a single integer. Then put the value 41 in the parameter list.
2. Create the direct object. In the Apple Event implementation of Web services, the direct object contains both the method name and the parameter list, as it is handed off to the subsystem that deals with the RPC calls. The direct object for a XML-RPC and a SOAP Web service call contains two fields: the `keyRPCMethodParam` field and the `keyRPCMethodName` field.
3. Create the event that you want to send. Just like in the normal Apple Event Manager, you create the target event, in this case an application URL type char, and then you provide the location that you are going to post the event to. You create the event with a special event class ID, `kAERPCClass`.
4. Send the event and deal with the reply. Apple events can be sent synchronously or asynchronously. If it is an asynchronous reply, you will see the reply as an Apple event.

In this example, you send the event, and then wait for the reply. This opens up the connection to the remote site, sends the event, gets the reply, and turns it back into an Apple event descriptor. You can then pull out of the reply the direct object, which is a string.

Create the Parameter List

You create the parameter list, which is a single integer.

```
AEDesc paramList;
AECreateList(NULL, 0, true, &paramList);
SInt32 ixState = 41;
AEPutPtr(&paramList, 0, typeSInt32, &ixState, sizeof(ixState));
```

Create the Direct Object for an XML-RPC Call

In this case, the direct object is a record.

```
AERecord directObject;
AECreateList(NULL, 0, true, &directObject);
```

The object has two fields:

```
AEPutParamDesc(&directObject, keyRPCMethodParam, &paramList)
            AEPutParamPtr(&directObject, keyRPCMethodName, typeChar, ?
            “examples.getStateName”, strlen(…))
```

You put the parameter list into the direct object. Note that this parameter list is a list of a single element.

```
AEList paramList;
AECreateList(NULL, 0, false, &paramList);
SInt32 ixState = 41;
AEPutPtr(&paramList, 0, typeSInt32, &ixState, sizeof(ixState));
AEPutParamDesc(&directObject, keyRPCMethodParam, &paramList);
AEDIsposeDesc(&paramList);
```

You put the method name to call in the direct object.

```
const char* methodName = "examples.getStateName";
AEPutParamPtr(&directObject, kRPCMethodName, typeChar, methodName,
strlen(methodName));
```

Create the Event

You describe the address (endpoint).

```
AEDesc addr;
AECreateDesc(typeApplicationURL, "http://betty.userland.com/RPC2",
                strlen(…), &addr)
```

You build the event.

```
AEDesc event;
AECreateAppleEvent(kAERPCClass, kAEXMLRPCScheme, &addr, …, &event);
AEPutParamDesc(&event, keyDirectObject, &directObject)
```

Send the Event

You send the event and then deal with the reply.

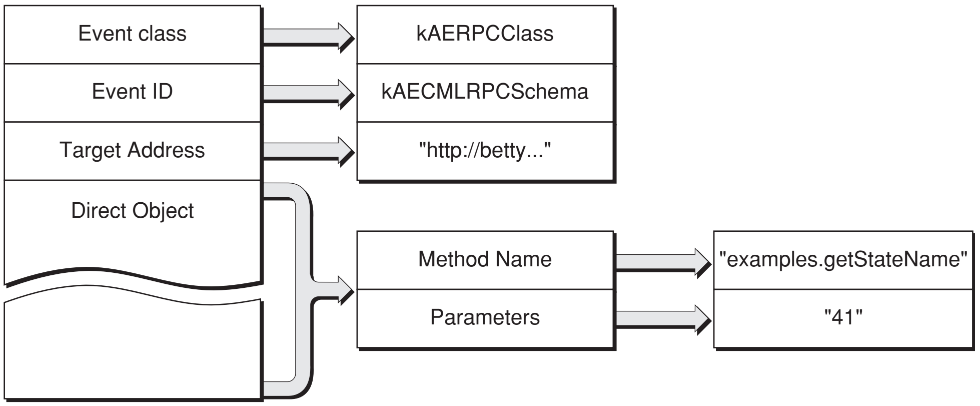
```
AEDesc reply;
AESend(&event, &reply, kAEWaitReply, …, …)

AEGetParamPtr(&reply, keyDirectObject, typeChar,
                  buffer, sizeof(buffer), …, &actualSize)
printf(“State: %.*s\n”, actualSize, buffer);
```

The method reply is contained within the direct object of the Apple event reply.

This sequence of actions is illustrated in Figure 2-1.

Figure 2-1 A sequence of actions used to target a website using Apple events



Using Web Services Core in Mac OS X version 10.2

Although the Apple event method described in the above example is useful and easy to implement, it may describe a process that is too high-level in the protocol stack, in the library chain of the system. Some developers may want a more advanced framework with greater, low-level functionality. Toward that end, Mac OS X version 10.2 introduces a new framework that implements WebServicesCore inside of Core Services.

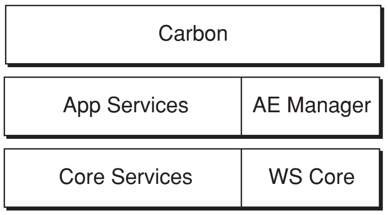
This is a low-level framework that sits alongside CFNetwork, Core Foundation and CarbonCore, as illustrated in Figure 2-2. This is a subframework, residing under the CoreServices umbrella, that lets you access the basic runtime model for the Mac OS X system. It is available to all applications, plugins, tools, and daemons.

The framework has no dependency on the Window Server or login window. You don't have to be a console user, so you can use it from any CGI application. It is fully integrated with the Mac OS X system, sitting inside Core Services, and leverages `CFXMLParser` and `CFNetwork`. The framework is thread-safe and based on the run loop.

It encourages you to asynchronously issue invocation requests on the run loop and receive a reply on your bundle. Because it is CFType-based, you have to create `CFType` objects for your strings, records, dictionaries and CF arrays. If you're an Objective-C programmer, you get "toll-free" bridging with Objective-C types. It's also a simple procedural API.

Figure 2-2 shows an illustration of WS Core sitting alongside Core Services in Mac OS X version 10.2.

Figure 2-2 WS Core alongside CoreServices in Mac OS X version 10.2



Low-Level Feature Set

The WebServicesCore API, which is described in Chapter 3, "Web Services Reference" (page 29), is protocol-independent—in other words, protocol-agnostic. To SOAP or XML-RPC, it looks the same. The difference is in what properties you set, using the `WSMethodInvocationRef` (page 47) object. You can have synchronous "one-shot" operation, or asynchronous operations with callbacks.

The framework is based on advances in CFNetwork, so as CFNetwork improves, WSCore will also improve. You get advanced networking support for firewalls and proxies. There is full access to the underlying CFHTTPMessage.

It is conceptually similar to scheduling in CFStream. This model is a little simpler, in that the synchronous calls give you a single callback with the result of the invocation.

The framework comes with HTTP / HTTPS support in 1.0. Future versions of will probably support other transport modes, as well.

WSMethodInvocationRef

The most useful object in the Web services core API is an opaque structure defined as `WSMethodInvocationRef` (page 47). This structure is created in the following steps:

1. You call a method inside the framework, giving it an endpoint URL and method name, and a prototype parameter which specifies the encoding style, that is, whether it is an XML-RPC or SOAP v1.1 method that you want to send.

2. You then add properties to the invocation reference that can control some of the serialization rules and transport options, such as the SOAP action or method namespace.
3. You set the parameters on the invocation as CFTypes. Then you can either schedule it on a run loop—or call it synchronously. It gives you back a result dictionary.

Creating the WSMethodInvocationRef Object

You create the `WSMethodInvocationRef` object by using the method `WSMethodInvocationCreate`, which takes the CF URL or NS URL. The method name is a `CFString`. The protocol is one of these CFString constants:

- `kWSXMLRPCProtocol`
- `kWSSOAP1999Protocol`

The header is Mac OS X version 10.2: `WSMethodInvocation.h`

WSMethodInvocation—Setting Parameters

Next, you set the parameters on the invocation. The call that you make, "WSMethodInvocationSetParameters" (page 33), takes the ref on which you are going to set the parameters. A dictionary containing the name of the parameters and the values of those parameters, and an array, which is optional for SOAP messages, specifies the order in which the parameters should be serialized. The array is not optional for XML-RPC method calls, because XML-RPC is positional—that is, it needs know the position. Some SOAP methods require that the order be specified by the service vendor as well. That is why you would want to have the second parameter of the array.

The `CFDictionary` contains the name of the parameter and a `CFType` for the parameter value, which can be one of the primitive CFTypes or a compound type. The types that are serialized automatically for you are `CFBoolean`, `CFNumber`, `CFDate`, `CFString`, `CFData`, `CFArray`, `CFDictionary`. You can also add meta data keys to a `CFDictionary` to specify custom namespaces, when it is serialized for SOAP, and the parameter order. You can build complex types using `CFDictionary`.

The types are serialized using "Section 5 Encoding" (page 13) SOAP encoding. The type information for custom types is not present in the invocation. If you have custom types, you have to write your own dictionary marshalling code, at least until a schema-aware WSDL parser is made available.

Building the Dictionary and Parameter Order Array

The following steps describe how you would build the CFdictionary and parameter order array.

1. You call `CFDictionaryAddValue` on a mutable dictionary that you've created:

   ```
   CFDictionaryAddValue(dict, CFSTR("param1"), CFSTR("Steve"
   CFDictionaryAddValue(dict, CFSTR("param2"), CFSTR("John"))
   CFArrayAddValue(order, CFSTR("param1"))
   CFArrayAddvalue(order, CFSTR("param2"))
   WSMethodInvocationSetParameters(ref, values, order)
   ```

2. Add the parameter names to the parameter name array.
3. Set the parameters onto the ref.
4. Invoke it, with a single call: `WSMethodInvocationInvoke`.

The result is a `CFDictionary`, which you must release. This is not the actual method results. The `CFDictionary` contains the method results, as well as optional debugging information from the request, the outgoing and returned XML, the HTTP headers and HTTP errors, so that if there was a networking transport error, that information will also be available in that dictionary.

The SOAP headers are returned as `kWSSOAPHeaderValues` `CFArray` of `CFStringRef`.

The faults may be manufactured for networking errors (`kWSNetworkStreamFaultString`).

WSMethodInvocation—Result Dictionary

You make this invocation call and if …

```
WSMethodResultIsFault(resultDict)
CFTypeRef faultString = CFDictionaryGetValue(result, kWSFaultString)
```

Also:

```
kWSFaultCode, kWSFaultExtra
```

Else, not a fault:

```
CFTypeRef myResult = CFDictionaryGetValue(?result,
kWSMethodInvocationResult)
```

The result of this is a `CFString`.

WSMethodInvocation—Asynchronous

1. Set the client (callback). This includes the info pointer that you want to pass to your callback, as well as some callbacks for the info pointer to allow it to be reference counted.
2. Once you've set the callback, you can schedule this invocation on one or more run loops. As those run loops execute, the invocation will pass through its state machine, which results in getting the data or getting a fault. When it completes, it will call your callback during that run loop invocation and you can process the result.
3. Callback invoked from the runloop when the invocation completes.
4. Can be scheduled on multiple runloops to implement thread groups.

WSDL Support

Currently the first release of Web Services does not include a WDSL API. A tool is available at `/Developer/Tools/WSMakeStubs` to generate static stubs. This tool parses the WSDL and produces templates for C++, Objective-C, and AppleScript. Various degrees of complexity are available in the generated stubs. The stub adheres to “Section 5” encoding and simple marshalling.

C H A P T E R 3

Web Services Reference

This chapter discusses the WebServicesCore framework available in Mac OS X version 10.2. The chapter describes the constants, data types and functions that comprise the framework.

The programming interface for the `WebServicesCore.framework` is declared in the following header file:

Mac OS X (10.2): WSMethodInvocation.h

Web Services Functions

The following APIs are provided in the `WebServicesCore.framework`:

- `WSMethodInvocationCreate` (page 30).
- `WSMethodInvocationCreateFromSerialization` (page 31)
- `WSMethodInvocationCopySerialization` (page 32).
- `WSMethodInvocationSetParameters` (page 33).
- `WSMethodInvocationCopyParameters` (page 33).
- `WSMethodInvocationSetProperty` (page 34).
- `WSMethodInvocationCopyProperty` (page 35).
- `WSMethodInvocationInvoke` (page 36).
- `WSMethodInvocationCallBackProcPtr` (page 37).

- `WSMethodInvocationSetCallBack` (page 38).
- `WSMethodInvocationScheduleWithRunLoop` (page 38).
- `WSMethodInvocationUnscheduleFromRunLoop` (page 39).
- `WSMethodResultIsFault` (page 40).
- `WSMethodInvocationSerializationProcPtr` (page 41).
- `WSMethodInvocationAddSerializationOverride` (page 42).
- `WSMethodInvocationDeserializationProcPtr` (page 43).
- `WSMethodInvocationAddDeserializationOverride` (page 43).
- `WSGetWSTypeIDFromCFType` (page 45).
- `WSGetCFTypeIDFromWSTypeID` (page 45).
- `WSMethodInvocationGetTypeID` (page 46).
- `WSMethodInvocationRef` (page 47).

Method Invocation Functions

The following is a list of method invocation functions provided in the `WebServicesCore.framework`.

WSMethodInvocationCreate

Creates a Web services method invocation object.

```
WSMethodInvocationRefWSMethodInvocationCreate(
        CFURLRef url,
        CFStringRef methodName
        CFStringRef protocol);
```

Parameter Descriptions

`url`

The endpoint of the service.

`methodName`

The name of the method to be called.

function result A `WSMethodInvocationRef` object that can be passed to `WSMethodInvocationInvoke` or scheduled with a run loop.

Discussion
This function creates a Web services method invocation object. This object may be executed synchronously or scheduled on a run loop for asynchronous execution.

Special Considerations
Mac OS X Threading

Thread safe

Availability
Mac OS X: in version 10.2 and later in WebServicesCore.framework

CarbonLib: not available

Non-Carbon CFM: not available

WSMethodInvocationCreateFromSerialization

Creates a Web services method invocation object from a previously serialized contract.

```
WSMethodInvocationRef WSMethodInvocationCreateFromSerialization
                               (CFDataRef contract);
```

Parameter Descriptions
`contract`
The result of a previously serialized `WSMethodInvocationRef`.

function result A `WSMethodInvocationRef` object that can be passed to `WSMethodInvocationInvoke` or scheduled with a run loop.

Discussion
This function creates a Web services method invocation object from a previously serialized contract.

Special Considerations
Mac OS X Threading

Thread safe

Availability
Mac OS X: in version 10.2 and later in WebServicesCore.framework

CarbonLib: not available

Non-Carbon CFM: not available

WSMethodInvocationCopySerialization

Creates a serialized version of the method invocation which can be reconstituted at a later time.

```
CFDataRef WSMethodInvocationCopySerialization(WSMethodInvocationRef
                                                        invocation);
```

Parameter Descriptions

`invocation`
: The invocation to serialize.

function result A `CFDataRef`.

Discussion
This function creates a serialized version of the method invocation which can be reconstituted at a later time.

Special Considerations
Mac OS X Threading

Thread safe

Availability
Mac OS X: in version 10.2 and later in WebServicesCore.framework

CarbonLib: not available

Non-Carbon CFM: not available

WSMethodInvocationSetParameters

Sets the parameters for a method invocation.

```
WSMethodInvocationSetParameters(
        WSMethodInvocationRef   invocation,
        CFDictionaryRef         parameters,
        CFArrayRef              parameterOrder);
```

Parameter Descriptions

`invocation`

The invocation object.

`parameters`

A `CFDictionaryRef` of `CFString` keys and `CFTypeRef` values.

`parameterOrder`

A `CFArrayRef` of `CFString` parameter names.

Discussion

This function sets the parameters for a method invocation. The `parameterOrder` may be NULL, in which case the order of the parameters is undefined. If it is not NULL and the parameters dictionary contains more parameters than are specified by the order, the behavior is undefined. If the `parameterOrder` specifies more parameters than are present in the dictionary, the result is undefined.

Special Considerations

Mac OS X Threading

Thread safe

Availability

Mac OS X: in version 10.2 and later in WebServicesCore.framework

CarbonLib: not available

WSMethodInvocationCopyParameters

Copies the parameters from the invocation.

```
CFDictionaryRef WSMethodInvocationCopyParameters(
        WSMethodInvocationRef   invocation,
        CFArrayRef *            parameterOrder);
```

Parameter Descriptions

`invocation`
: The neighborhood that is to be copied.

`parameterOrder`
: A pointer to a `CFArray` which will will receive the names, in their specified order, of the input parameter values. This parameter may be NULL.

Discussion

This function copies the parameters from the invocation. The resulting dictionary contains the parameter dictionary. The `parameterOrder` output parameter, if not NULL, will contain the order used to serialize the parameters.

Special Considerations

Mac OS X Threading

Thread safe

Availability

Mac OS X: in version 10.2 and later in WebServicesCore.framework

CarbonLib: not available

Non-Carbon CFM: not available

WSMethodInvocationSetProperty

Adds “properties” to a method invocation.

```
void WSMethodInvocationSetProperty(
       WSMethodInvocationRef   invocation,
       CFStringRef             propertyName,
       CFTypeRef               propertyValue);
```

Parameter Descriptions

`invocation`
: The invocation.

`propertyName`
: A `CFStringRef` name of the property to modify.

`propertyValue`
: A `CFTypeRef` containing the new property value

function result None.

Discussion

This function adds “properties” to a method invocation. These properties can be user-defined or one of the WebServicesCore declared properties (which may modify the behavior of the invocation.) All WebServicesCore declared properties will start with the string `"kWS"`, for example, `kWSHTTPFollowsRedirects`.

Properties are serialized along with the contract, so you may want to avoid sticking raw pointers in a `CFNumber`, for example.

Special Considerations

Mac OS X Threading

Thread safe

Availability

Mac OS X: in version 10.2 and later in WebServicesCore.framework

CarbonLib: not available

Non-Carbon CFM: not available

`WSMethodInvocationCopyProperty`

Returns a property from a invocation.

```
CFTypeRef WSMethodInvocationCopyProperty(
       WSMethodInvocationRef   invocation,
       CFStringRef             propertyName) ;
```

Parameter Descriptions

`invocation`
: The invocation.

`propertyName`
: The name of the property to retrieve.

Discussion

This function returns a property from a invocation. If the result is NULL, the property doesn’t exist. Because this is a Copy call, you must release the result.

Special Considerations

Mac OS X Threading

Thread safe

Availability
Mac OS X: in version 10.2 and later in WebServicesCore.framework

CarbonLib: not available

Non-Carbon CFM: not available

WSMethodInvocationInvoke

Executes the invocation.

```
CFDictionaryRef WSMethodInvocationInvoke(WSMethodInvocationRef
                                                    invocation) ;
```

Parameter Descriptions

`invocation`
: The invocation.

function result A `CFDictionaryRef` containing the result of the execution or a fault, and optional debug information.

Discussion
This function executes the invocation. If the call was successful, the result will contain the result of the invocation. If, for some reason, the invocation fails, including out of memory or invalid parameter errors, then the result will contain a fault structure. You must release the result when you're done with it.

Special Considerations
Mac OS X Threading

Thread safe

Availability
Mac OS X: in version 10.2 and later in WebServicesCore.framework

CarbonLib: not available

Non-Carbon CFM: not available

Callback Functions

The Web services calls described in this section implement the asynchronous variant of the WSMethodInvocationInvoke (page 36) routine.

The strategy when using these calls is to schedule the invocation on a given run loop. When the invocation completes, it calls the specified callback with the result of the execution. The callback is responsible for releasing the result ref.

Your code is responsible for unscheduling the invocation from the run loop, whether it completes or not. You can re-schedule an invocation after it completes. When you unschedule with the run loop, the CallBack is not called.

If a network error occurs, the kWSFaultString entry of the result will contain some textual description of the error, kWSFaultCode will contain kWSNetworkingFault and kWSFaultExtra will be a dictionary containing two CFNumber values called kWSStreamErrorDomain and kWSStreamErrorError.

WSMethodInvocationCallBackProcPtr

Prototypes the callback made when an asynchronous invocation completes.

```
typedef CALLBACK_API( void , WSMethodInvocationCallBackProcPtr
   )(WSMethodInvocationRef invocation, void *info, CFDictionaryRef outRef);
```

Parameter Descriptions

invocation	The invocation just completed.
info	Private callback data.
outRef	A CFDictionaryRef containing the result of the execution or a fault, and optional debug information.

Discussion

This callback function prototypes the callback made when an asynchronous invocation completes. This callback is passed a reference to the invocation just completed, a pointer to private data, and a dictionary that contains the return value or falut for this invocation. The callback is responsible for releasing the dictionary when it is no longer used.

WSMethodInvocationSetCallBack

Sets the callback for an asynchronous method invocation.

```
extern void
   WSMethodInvocationSetCallBack(
       WSMethodInvocationRef invocation,
       WSMethodInvocationCallBackProcPtr clientCB,
       WSClientContext * context);
```

Parameter Descriptions

`invocation`	The invocation.
`clientCB`	A `ProcPtr` to be called when the invocation completes.
`context`	A pointer to a `WSClientContext`. The structure will be copied.

Discussion

This function sets the callback for an asynchronous method invocation. You call this with a `clientCB` and `context` of NULL to clear the invocation callback.

Special Considerations

Mac OS X Threading

Thread safe

Availability

Mac OS X: in version 10.2 and later in WebServicesCore.framework

CarbonLib: not available

Non-Carbon CFM: not available

WSMethodInvocationScheduleWithRunLoop

Schedules the invocation to execute on the run loop.

```
extern void
   WSMethodInvocationScheduleWithRunLoop(
     WSMethodInvocationRef   invocation,
     CFRunLoopRef            runLoop,
     CFStringRef             runLoopMode);
```

Parameter Descriptions

`invocation`
: The invocation.

`runLoop`
: The run loop upon which to scheduile the invocation.

`runLoopMode`
: The run loop mode.

Discussion

This function schedules the invocation to execute on the run loop.

Special Considerations

Mac OS X Threading

Thread safe

Availability

Mac OS X: in version 10.2 and later in WebServicesCore.framework

CarbonLib: not available

Non-Carbon CFM: not available

WSMethodInvocationUnscheduleFromRunLoop

Unschedules the invocation from a given run loop and mode.

```
extern void
   WSMethodInvocationUnscheduleFromRunLoop(
     WSMethodInvocationRef   invocation,
     CFRunLoopRef            runLoop,
     CFStringRef             runLoopMode);
```

Parameter Descriptions

`invocation`
: The invocation.

`runLoop`
: The run loop upon which to scheduile the invocation.

`runLoopMode`
: The run loop mode.

Discussion
This function unschedules the invocation from a given run loop and mode. If the invocation has not yet completed, its callback will not be called.

Special Considerations
Mac OS X Threading

Thread safe

Availability
Mac OS X: in version 10.2 and later in WebServicesCore.framework

CarbonLib: not available

Non-Carbon CFM: not available

WSMethodResultIsFault

Returns TRUE if the method invocation result contains a fault.

```
extern Boolean
       WSMethodResultIsFault(CFDictionaryRef methodResult);
```

Parameter Descriptions
`methodResult`
The result ref.

`This function returns TRUE if the method invocation result contains a fault.`

Discussion
Result interrogation: If the result is a fault, look in the `kWSFaultCode`, `kWSFaultString` and `kWSFaultExtra` fields of the resulting dictionary. If not a fault, `kWSMethodInvocationResult` will contain the result of the execution. If debugging information was requested, it will be available in the dictionary as well.

Special Considerations
Mac OS X Threading

Thread safe

Availability
Mac OS X: in version 10.2 and later in WebServicesCore.framework

CarbonLib: not available

Non-Carbon CFM: not available

Serialization and Deserialization Override Support

You can add serialization and deserialization callbacks for custom types, or types not otherwise handled by the framework, as discussed in this section. Note that these properties are not serialized if the invocation is serialized.

WSMethodInvocationSerializationProcPtr

Prototypes the callback function for a custom serialization procedure.

```
typedef CALLBACK_API( CFStringRef , WSMethodInvocationSerializationProcPtr
   )(WSMethodInvocationRef invocation, CFTypeRef obj, void *info);
```

Parameter Descriptions

`invocation`
: The invocation currently being serialized.

`obj`
: The `CFTypeRef` to be serialized.

`info`
: Private callback data.

A `CFStringRef` containing valid XML. The caller of this callback will release the string.

Discussion

This function prototypes the callback function for a custom serialization procedure. This callback is called whenever a type has the given CFTypeID. The callback should return an XML snippet that will be “understood” by the server as a correct serialization for a given type. If the callback returns NULL, the default serializer will be used.

WSMethodInvocationAddSerializationOverride

Specifies a callback which will be called to produce the XML that represents the serialization of a given type ref.

```
extern void
   WSMethodInvocationAddSerializationOverride(
     WSMethodInvocationRef                     invocation,
     CFTypeID                                  objType,
     WSMethodInvocationSerializationProcPtr    serializationProc,
     WSClientContext *                         context);
```

Parameter Descriptions

`invocation`
: The invocation.

`objType`
: The `CFTypeID` of the object.

`serializationProc`
: The callback called.

`context`
: A pointer to a `WSClientContext`. The structure will be copied.

Discussion

This function specifies a callback which will be called to produce the XML that represents the serialization of a given type ref. See `WSDescription.h` for a list of CFTypes for which there currently exist serializers. If your callback returns NULL, the default serializer will be used.

Special Considerations

Mac OS X Threading

Thread safe

Availability

Mac OS X: in version 10.2 and later in WebServicesCore.framework

CarbonLib: not available

Non-Carbon CFM: not available

WSMethodInvocationDeserializationProcPtr

Prototypes the callback function for a custom deserializer.

```
typedef CALLBACK_API( CFTypeRef , WSMethodInvocationDeserializationProcPtr
   )(WSMethodInvocationRef invocation, CFXMLTreeRef msgRoot, CFXMLTreeRef
   deserializeRoot, void *info);
```

Parameter Descriptions

`invocation`
The invocation executing.

`msgRoot`
The root tree element.

`deserializeRoot`
The tree element that needs to be deserialized.

`info`
Private callback data.

function result A `CFTypeRef` representing the deserialized data, or NULL to allow the default deserializers to act.

Discussion

This function prototypes the callback function for a custom deserializer. The callback is passed a reference to the invocation currently being executed, the root of the response parse tree, the current node being deserialized, and a pointer to private data. The return result should be a valid `CFTypeRef` object (which will be released by the caller) or NULL to allow the default deserializer to act.

WSMethodInvocationAddDeserializationOverride

Specifies a callback to be made when parsing an XML method response.

```
extern void
   WSMethodInvocationAddDeserializationOverride(
     WSMethodInvocationRef                     invocation,
     CFStringRef                               typeNamespace,
     CFStringRef                               typeName,
     WSMethodInvocationDeserializationProcPtr  deserializationProc,
     WSClientContext *                         context);
```

Parameter Descriptions

`invocation`
: The invocation.

`typeNamespace`
: The fully resolved namespace for a specific type. If NULL, the default namespace will be used. For example, this field could be: CFSTR("http://www.w3.org/2001/XMLSchema-instance").

`typeName`
: The non-qualified type name. This parameter must not be NULL.

`deserializationProc`
: A `ProcPtr` to be called to perform the deserialization.

`context`
: A pointer to a `WSClientContext`. The structure will be copied.

Discussion

This function specifies a callback to be made when parsing an XML method response. The callback should return a `CFTypeRef` containing the deserialized object value. If the callback returns NULL, the default deserializer will be used.

Special Considerations

Mac OS X Threading

Thread safe

Availability

Mac OS X: in version 10.2 and later in WebServicesCore.framework

CarbonLib: not available

Non-Carbon CFM: not available

Other Functions

The following is a list of other functions and their parameter descriptions.

WSGetWSTypeIDFromCFType

Returns the `WSTypeID` associated with `CFTypeRef`.

```
extern WSTypeID
       WSGetWSTypeIDFromCFType(CFTypeRef ref);
```

Parameter Descriptions

`ref`	A `CFTypeRef` object.
	The `WSTypeID` used in serializing the object. If no `WSTypeID` matches, `eWSUnknownType` is returned.

Discussion

This function returns the `WSTypeID` associated with CFTypeRef. There is not a one-to-one mapping between `CFTypeID` and `WSTypesID`. Therefore, an actual instance of a `CFType` must be passed.

Special Considerations

Mac OS X Threading

Thread safe

Availability

Mac OS X: in version 10.2 and later in WebServicesCore.framework

CarbonLib: not available

Non-Carbon CFM: not available

WSGetCFTypeIDFromWSTypeID

Returns the `CFTypeID` that is associated with a given `WSTypeID`.

```
extern CFTypeID
       WSGetCFTypeIDFromWSTypeID(WSTypeID typeID);
```

Parameter Descriptions

`typeID`	A `WSTypeID` constant.
function result	A CFTypeID, or 0 if not found.

Discussion

This function returns the `CFTypeID` that is associated with a given `WSTypeID`. CFTypeIDs are only valid during a particular instance of a process and should not be used as static values.

```
typedef CALLBACK_API( void *, WSClientContextRetainCallBackProcPtr )(void *
info);
typedef CALLBACK_API( void , WSClientContextReleaseCallBackProcPtr )(void *
info);
typedef CALLBACK_API( CFStringRef ,
WSClientContextCopyDescriptionCallBackProcPtr )(void * info);
```

WSMethodInvocationGetTypeID

```
extern CFTypeID
       WSMethodInvocationGetTypeID(void);
```

Discussion

Special Considerations

Mac OS X Threading

Thread safe

Availability

Mac OS X: in version 10.2 and later in WebServicesCore.framework

CarbonLib: not available

Non-Carbon CFM: not available

Web Services Data Types

WSClientContext

Several calls in `WebServicesCore.h` take a callback with an optional `context` pointer. The `context` is copied and the `info` pointer retained. When the callback is made, the `info` pointer is passed to the callback.

```
struct WSClientContext {
    CFIndex             version;
    void *              info;
    WSClientContextRetainCallBackProcPtr  retain;
    WSClientContextReleaseCallBackProcPtr  release;
    WSClientContextCopyDescriptionCallBackProcPtr  copyDescription;
};
typedef struct WSClientContext          WSClientContext;
```

Field Descriptions

`version`
: Set to zero.

`info`
: Info pointer to be passed to the callback

`retain`
: Callback made on the info pointer. This field may be NULL.

`release`
: Callback made on the info pointer. This field may be NULL.

`copyDescription`
: Callback made on the info pointer. This field may be NULL.

WSMethodInvocationRef

A `WSMethodInvocationRef` represents an object that can be executed to obtain a result from a Web service. This is a `CFType` and is therefore reference counted and and should be managed via `CFRetain` and `CFRelease`.

```
typedef struct OpaqueWSMethodInvocationRef*  WSMethodInvocationRef;
```

Web Services Constants

Internally, WebServicesCore uses the following enumeration when serializing between CoreFoundation and XML types. Because CFTypes are defined at runtime, it is not always possible to produce a static mapping to a particular `CFTypeRef`. This enumeration and the associated API allows for static determination of the expected serialization.

Web Service Protocol Types

These constant strings specify the type of Web service method invocation created. These are passed to `WSMethodInvocationCreate` (page 30).

```
extern CFStringRef kWSXMLRPCProtocol;
extern CFStringRef kWSSOAP1999Protocol;
extern CFStringRef kWSSOAP2001Protocol;
```

Discussion
For more information on these service types, refer:

- XML-RPC: <http://www.xml-rpc.com/spec/>
- SOAP 1.1: <http://www.w3.org/TR/SOAP/>
- SOAP 1.2: <http://www.w3.org/2002/ws/>

Dictionary Entries

Dictionary entry, if the invocation result is not a fault, is always available in method responses. For SOAP messages, though, it may be more correct to query the result dictionary for the specific field that you are interested in.

What this really means is that the dictionary returned by the invocation may contain more than one value, wherein the result data is duplicated several times. If you don’t know what to ask for to dump the reply, you can ask for this key. If you do know what you want, you should request that field explicitly.

You can also specify the name of the reply parameter in the invocation, using kWSMethodInvocationResultParameterName. This adds an alias for the given name to the result dictionary, so that kWSMethodInvocationResult will always return the correct parameter. Note that this will not work for multi-value returns.

```
extern CFStringRef kWSMethodInvocationResult;
```

Dictionary entries if the result is a fault:

```
extern CFStringRef kWSFaultString;  /* a CFString */
extern CFStringRef kWSFaultCode; /* a CFNumber */
extern CFStringRef kWSFaultExtra;   /* a CFString or CFDictionary, or
                                          NULL */
```

If the result is a fault, and if the value of kWSFaultString in the reply dictionary is kWSNetworkStreamFaultString, then kWSFaultExtra will be a dictionary indicating the network error and kWSFaultCode is ignored in this case.

Refer to <CoreFoundation/CFStream.h> for details on what the domain and error numbers mean.

```
extern CFStringRef kWSNetworkStreamFaultString;
extern CFStringRef kWSStreamErrorMessage; /* A CFString (for debug
                                                      purposes only) */
extern CFStringRef kWSStreamErrorDomain; /* A CFNumberRef */
extern CFStringRef kWSStreamErrorError; /* A CFNumberRef */
```

Specifying CFHTTPMessageRef as a Property

For HTTP[S] based invocations, you can specify a CFHTTPMessageRef as a property which will be used instead of creating a new outgoing message. The CFHTTPMessageRef can contain header, proxy and authentication information. The body of the message will be ignored and replaced with the outgoing, serialized invocation.

After the invocation has executed, you can retrieve a copy of the actual CFHTTPMessageRef, containing the details of the invocation using kWSHTTPResponseMessage. Attempting to retrieve the response message property before the invocation completes will result return NULL.

Refer to <CFNetwork/CFHTTPMessage.h> for more information.

```
extern CFStringRef kWSHTTPMessage; /* CFHTTPMessageRef */
extern CFStringRef kWSHTTPResponseMessage; /* CFHTTPMessageRef
```

To avoid having to create an entire `CFHTTPMessageRef`, these properties are individually settable. If they are set, they will override any `CFHTTPMessageRef` previously specified.

```
extern CFStringRef kWSHTTPVersion; /* "http/1.1" */
extern CFStringRef kWSHTTPExtraHeaders; /* a CFDictionary of { key (CFString),
val (CFString) } pairs */
extern CFStringRef kWSHTTPProxy; /* CFURfRefL */
extern CFStringRef kWSHTTPFollowsRedirects; /* kCFBooleanFalse */
```

SOCKS Proxy Support

`WSMethodInvocation` uses the same flags as `CFSocketStream.h` in configuring SOCKS proxy support. You can set the `kCFStreamPropertySOCKSProxy` property on the invocation and the value will be applied to the underlying stream. See `CFSocketStream.h` for more information and valid keys.

```
extern CFStringRef kWSDebugOutgoingHeaders; /* kCFBooleanFalse */
extern CFStringRef kWSDebugOutgoingBody; /* kCFBooleanFalse */
extern CFStringRef kWSDebugIncomingHeaders; /* kCFBooleanFalse */
extern CFStringRef kWSDebugIncomingBody; /* kCFBooleanFalse */
```

Discussion

These debugging flags will populate the `WSInvocationResultRef` with some potentially useful debugging output. The property name of the flag is the same as the the field in the result dictionary.

Extra Properties for SOAP Messages

These apply to the message namespace and format itself. Individual message elements can be modified using the `kWSRecord` constants below.

```
extern CFStringRef kWSSOAPMethodNamespaceURI;   /* CFStringRef */
extern CFStringRef kWSSOAPBodyEncodingStyle;    /* CFStringRef {
                                  kWSSOAPStyleDoc, kWSSOAPStyleRPC } */
extern CFStringRef kWSSOAPStyleDoc;
extern CFStringRef kWSSOAPStyleRPC;
```

For SOAP messages, this is an array of CFStringRefs which contain valid XML header elements that are sent with the message. These are only applicable to the header of a SOAP message.

```
extern CFStringRef kWSSOAPMessageHeaders; /* CFArrayRef */
```

When serializing a record (dictionary), these keys present in the dictionary can modify the behavior of the serialization.

```
extern CFStringRef kWSRecordParameterOrder;/* CFArrayRef of CFStringRef
                                                          */
extern CFStringRef kWSRecordNamespaceURI; /* CFStringRef */
extern CFStringRef kWSRecordType; /* CFStringRef */
```

Specifies that the result parameter will be found as this name. This forces the deserializer to alias the named output parameter to `kWSMethodInvocationResult`

```
extern CFStringRef kWSMethodInvocationResultParameterName;
```

Specifies a timeout (as `CFNumber`) which specifies in seconds the amount of time to wait for the invocation to complete. If the invocation times out before the server results are returned, a fault will be returned with the error code `errWSTimeoutError`.

```
extern CFStringRef  kWSMethodInvocationTimeoutValue;
```

WSTypeID

```
enum WSTypeID {
eWSUnknownType              = 0,
eWSNullType                 = 1,
eWSBooleanType              = 2,
eWSIntegerType              = 3,
eWSDoubleType               = 4,
eWSStringType               = 5,
eWSDateType                 = 6,
eWSDataType                 = 7,
eWSArrayType                = 8,
eWSDictionaryType           = 9
};
typedef enum WSTypeID WSTypeID;
```

Constant Descriptions

`eWSUnknownType`
: No mapping is known for this type.

`eWSNullType`
: CFNullRef

`eWSBooleanType`
: CFBooleanRef.

`eWSIntegerType`
: CFNumberRef for 8, 16, and 32-bit integers.

`eWSDoubleType`
: CFNumberRef for long double real numbers.

`eWSStringType`
: CFStringRef.

`eWSDateType`
: CFDataRef.

`eWSArrayType`
: CFArrayRef.

`eWSDictionaryType`
: CFDictionaryRef.

Web Services Result Codes

The result codes specific to Web services are listed here. In addition, Web services functions may return other Mac OS X result codes, which are described in Inside Mac OS X.

`errWSInternalError`	-65793L	An internal framework error.
`errWSTransportError`	-65794L	A network error occured.
`errWSParseError`	-65795L	The server response wasn't valid XML.
`errWSTimeoutError`	-65796L	The invocation timed out.

Document Revision History

The following is a change log of this document.

Table A-1 Web services document revision history

Version	Notes
Oct. 10, 2002	Replaced and updated description of XML-RPC vs. SOAP.
Sept. 19, 2002	Added fixes and corrections from Apple Engineering to the alpha draft. Updated cross-references; converted document to structured FrameMaker format. Tested HTML version. Released to Production for publication on Apple’s developer documentation Website under the category of Networking.
Aug. 24, 2002	Preliminary alpha draft completed for engineering review. Draft discusses Web services features, programming tasks, and APIs available in Mac OS X (10.2).